Investigation of Forward Error Correction Coding Schemes for a Broadcast Communication System

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science
in
Computer Science
by
Xiaohan (Sasha) Wang

Supervisor: Dr Andreas Willig
Co-supervisor: Dr Graeme Woodward

University of Canterbury
March 2013

To the 365 days I spent in NZi3.

Abstract

This thesis investigates four FEC (forward error correction) coding schemes for their suitability for a broadcast system where there is one energy-rich transmitter and many energy-constrained receivers with a variety of channel conditions. The four coding schemes are: repetition codes (the baseline scheme); Reed-Solomon (RS) codes; Luby-Transform (LT) codes; and a type of RS and LT concatenated codes. The schemes were tested in terms of their ability to achieve both high average data reception success probability and short data reception time at the receivers (due to limited energy). The code rate (R_c) is fixed to either 1/2 or 1/3. Two statistical channel models were employed: the memoryless channel and the Gilbert-Elliott channel. The investigation considered only the data-link layer behaviour of the schemes. During the course of the investigation, an improvement to the original LT encoding process was made, the name LTAM (LT codes with Added Memory) was given to this improved coding method. LTAM codes reduce the overhead needed for decoding short-length messages. The improvement can be seen for decoding up to 10000 number of user packets. The maximum overhead reduction is as much as 10% over the original LT codes.

The LT-type codes were found to have the property that can both achieve high success data reception performance and flexible switch off time for the receivers. They are also adaptable to different channel characteristics. Therefore it is a prototype of the ideal coding scheme that this project is looking for. This scheme was then further developed by applying an RS code as an inner code to further improve the success probability of packet reception. The results show that LT&RS code has a significant improvement in the channel error tolerance over that of the LT codes without an RS code applied. The trade-off is slightly more reception time needed and more decoding complexity. This LT&RS code is then determined to be the best scheme that fulfils the aim in the context of this project which is to find a coding scheme that both has a high overall data reception probability and short overall data reception time.

Comparing the LT&RS code with the baseline repetition code, the improvement is in three aspects. Firstly, the LT&RS code can keep full success rate over channels have approximately two orders of magnitude more errors than the repetition code. This is for the two channel models and two code rates tested. Secondly, the LT&RS code shows an exceptionally good performance under burst error channels. It is able to maintain more than 70% success rate under the long burst error channels where both the repetition code and the RS code have almost zero success probability. Thirdly, while the success rates are improved, the data reception time, measured in terms of number of packets needed to be received at the receiver, of the LT&RS codes can reach a

maximum of 58% reduction for $R_c = 1/2$ and 158% reduction for $R_c = 1/3$ compared with both the repetition code and the RS code at the worst channel error rate that the LT&RS code maintains almost 100% success probability.

Acknowledgments

First of all, I would like to express my sincere gratitude to my senior supervisor Dr Andreas Willig and my co-supervisor Dr Graeme Woodward. Andreas was the person that first suggested this project to me. I have built up my confidence in research work through numerous conversations with him. At the same time, he also encouraged me to reach for higher goals and research quality. I am grateful for that. The same thank you also goes to Dr Graeme Woodward. Despite being busy, he always gave the quickest response to my questions and issues related to the project. As a research leader, he has always been a strong back up to me as well as the whole research team. I thank both of my supervisors for the valuable guidance and immense support throughout this project. Secondly, I would like to thank the New Zealand Ministry of Science and Innovation (MSI) for providing the funds for this project and the collaborating industrial partner Connexionz for providing relevant application details for setting up the system model of the project. Last but not least, I thank my parents for their understanding and support towards my studies which gave me the advantage of having much more time and concentration for accomplishing this project.

Table of Contents

List of Tables

List of Figures

List of Symbols

α Confidence level

μ Probability mass function of the Robust Soliton distribution

ρ Probability mass function of the Ideal Soliton distribution

BER_b Bit error rate of the bad channel in the Gilbert-Elliott channel model

BER_g Bit error rate of the good channel in the Gilbert-Elliott channel model

d Degree of an encoded packet used in the LT codes

E Packet erasure rate of a binary erasure channel

e Bit error rate of a binary symmetric channel

E_b/N_0 Energy per bit to noise power spectral density ratio

j Number of bits per packet

k The total number of user data packet

k_{RS} Number of user data symbols contained in an RS encoded packet

m Symbol size of an RS code

n Depend on the context, this symbol is used to denote several things: degree of freedom of the Student-T distribution in the system model chapter, the total number of LT encoded packets in the LT code chapter and the number of received packets needed for decoding in the performance comparison chapter.

n' Number of received LT packets

n_{RS} Total number of symbols in an RS encoded packet

$p_{b,g}$	Transition probability from bad state to good state in a Gilbert-Elliott channel model
$p_{g,b}$	Transition probability from good state to bad state in a Gilbert-Elliott channel model
p_{rep}	Bit success probability in the binomial experiment in the repetition coding scheme
P_{suc}	Packet success probability
R_c	Code rate
X	Number of repetitions for the repetition code, reciprocal of R_c

Chapter I

Introduction

Forward Error Correction (FEC) Coding, also known as Error Control Coding (ECC) is often used for data transmissions over unreliable channels. It improves the efficiency and reliability of data transmissions [21] by adding redundant information to the user data[1]. There is a wide variety of FEC codes. They are mainly distinguished by the encoding and decoding algorithms and error correcting capabilities. There is usually a trade-off between the encoding and decoding complexities of a code and its error correcting capability. Therefore the selection of the FEC code for a system should be carefully examined according to the system requirements.

1.1 Background

At the data-link layer of communication systems, both FEC and ARQ (Automatic Repeat reQuest) can be used for ensuring the delivery of information. However, feedback channels for ARQ are not always possible. For example, when receivers are not equipped with the transmitting function. In cases like broadcast and multicast transmissions, feedback channels not only cause feedback implosion[5] which can quickly overwhelm the transmitter, but also the data segment losses of the receivers tends to be uncorrelated which greatly increases the retransmission load [37]. Therefore ARQ is also not suitable. In these situations, FEC becomes more important for ensuring the reliability of data transmission.

In 1948, a seminal paper published by Shannon [40] established and formalised the field of information theory. In this paper, Shannon stated that message can be transmitted with arbitrarily small errors with a random coding technique up to the capacity (measured as the information rate) of the channel. From that time, researchers have sought error correction codes capable of approaching this information rate. The challenges to this include keeping the code with practical complexity and finite length.

[1] The information at one place that is desired to be reproduced at another place.

There are two types of coding techniques: source coding and channel coding. Source coding eliminates unnecessary data contained in the user data, and channel coding adds redundant information to the user data so there is a possibility that erroneous data can be corrected after they have passed an unreliable channel. The focus of this thesis is on the channel coding part, which is a form of FEC.

1.2 Problem Definition

The broadcast transmission system to be considered in this study has been generalised from an industrial-based communication application. It is a wireless broadcast system. The system contains one transmitter and a large number of receivers with different channel characteristics. The transmitter is energy-rich and computing resource rich. The receivers have limited energy and computational resources. The broadcast message is finite and it is in the order of kilobytes or tens of kilobytes.

After studying the communication process of the broadcast system, it has been found that the FEC used by the application could be improved in order to increase the overall probability of successful data reception. The currently used FEC is repetition code; it has a very poor error-correcting capability. The motivation was therefore to improve the performance of the system by determining a better suited FEC scheme. The improvements are focused on two aspects: the probability of successful data reception[2] and the average data reception time[3].

The aim of this thesis is therefore to find a better FEC coding scheme for a broadcast system that can achieve both high overall successful data reception probability and low overall data reception time for the receivers with a range of channel conditions. Three candidate coding schemes were initially proposed to be investigated for their suitability for the considered broadcast system. They are repetition codes – the baseline coding scheme, Reed-Solomon (RS) codes – a traditional error correction code with a strong error-correcting capability, and Luby Transform (LT) codes – a type of fountain codes that are specifically designed for broadcast systems. Another two schemes have been derived during the course of the investigation: an improved version of the LT code called LTAM (LT codes with Added Memory) codes and a combination of RS and LTAM code.

The coding schemes are applied to the data-link layer of the communication hierarchy. No physical layer modelling of the transmitter, the receiver, or the channel is involved. User data are

[2] The average probability that the entire user message is successfully decoded at the receiver.

[3] The shorter this is, the less energy consumption for the receivers which is desired for energy limited receivers.

assumed to be packetised. Figure 1.1 shows a block diagram of the considered communication process. A more detailed system model is described in Chapter 2.

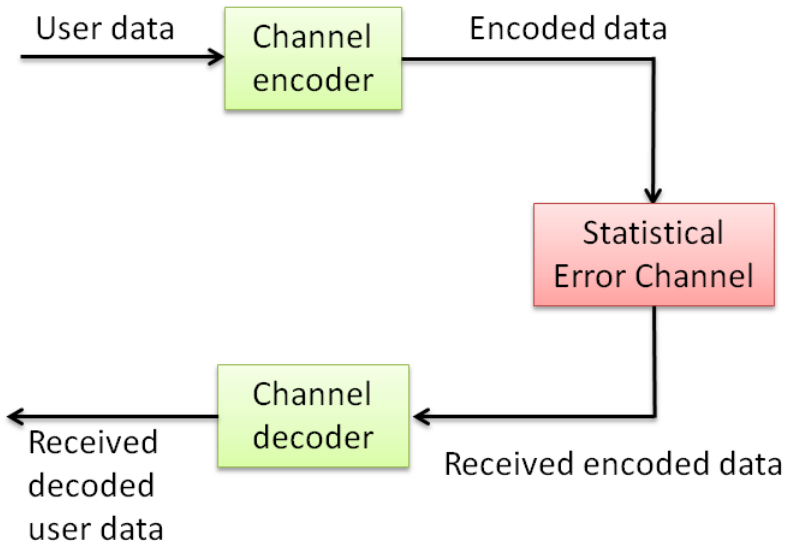

Figure 1.1: The communication system to be studied.

1.3 Selection of Candidate Schemes

Because of the limited time frame, the intention of this project is not to find an optimal coding scheme for the considered broadcast system. The main criterion for selecting the candidate coding scheme is the decoding complexity. This must be relatively low because the receivers do not have high computational power and they are energy-constrained. The acceptable complexity is polynomial with respect to the number of user packets and the packet size[4]. The selection of the three candidate schemes was based on this criterion as well as some other attractive features of the code. Repetition codes were selected as the baseline scheme because they are the scheme that is currently used in the focused application; this enables performance comparison to the scheme used in the current system. RS codes were selected due to their strong error correcting ability at relatively low complexity. LT codes were selected because they have the "rateless" feature that is particularly suitable for broadcast transmissions.

[4] Only the asymptotic complexity of the codes were considered. There were no considerations about the detailed complexity of one specific implementation.

1.4 Contributions

To the author's knowledge, this work is the first to compare the performance of the recently developed rateless fountain codes with intra-packet RS codes, as well as repetition codes specifically for broadcast communication systems in terms of success rate and data reception time. The main contributions of this work are:

- The novel development of an improved version of LT codes, the LTAM codes. These give a noticeable reduction in the amount of received data required for decoding for small message lengths. A paper has been published [47] based on the improvement made. Details are described in Section 5.3.

- Implementation of the simulations of the candidate coding schemes.

- Concatenation of the LTAM and RS codes which gives outstanding performance in channel error tolerance and data reception time.

- Performance comparison among all the candidate coding schemes.

1.5 Remaining Chapters

The remaining content of the thesis is organised as follows: Chapter 2 presents the system model. Chapters 3, 4 and 5 introduce the three proposed coding schemes separately. Each chapter describes the implementation and validation of each coding scheme, and explains decisions made on choosing various parameters in order to produce the best results with graphical support. Chapter 5 also presents the LTAM codes. The comparisons among all coding schemes are then presented in Chapter 6 Performance Comparison. This is followed by related work and future work (Chapter 7), and conclusions (Chapter 8). There are four appendices. Appendix A is the glossary, Appendix B presents additional results for the LTAM codes, and Appendix C gives a list of statistical significance for most graphical simulation results presented in this thesis. Appendix D presents the source code of the LTAM encoding and decoding MATLAB implementation.

Chapter II

System Model

This project involves identifying candidate coding schemes, determining performance measures, building and validating simulation models, executing simulations and comparing results. This comparisons are based on MATLAB simulations. This chapter describes the system model used for the simulations, the performance measures and the top level simulation set-up.

2.1 System Overview

The system model consists of one transmitter, one receiver and a wireless channel in between[1]. The user data consists of k packets, and the transmitter has a fixed transmission budget of $X \cdot k$ packets, where $X \in \{2, 3\}$. The receiver is switched on at the time the data transfer commences and switches itself off when it either has decoded all k user data packets or when all $X \cdot k$ packets have been exhausted. All performance measures defined below consider only this transmission process. Below is a list of assumptions:

- Receivers do not have relay functions; receivers receive data only from the transmitter and not from any neighbouring receivers.
- User data are finite and are partitioned into k packets. All packets are assumed to have the same size.
- Each packet is equipped with a checksum and it is assumed to be perfect when a packet level erasure channel is used. It detects errors reliably.
- Packet overhead[2] is ignored.

[1] The actual system has many receivers, but the system model built in this project uses one receiver with a variety of channel realisations to reflect different receivers in a broadcast system.

[2] control information contained in a transmitted packet

2.2 Data Structure

The size of the user data is approximately 4kB and the maximum packet size is 62 bytes (or 496 bits). These are determined according to the reference application. Since the RS code is the only scheme among the candidate coding schemes that has restrictions on packet sizes, in order to make consistent comparisons among all the candidate schemes the packet size used for all coding schemes is set to be the same as that for the RS code[3]. The total user data size is set to be 32384 bits, or 4048 bytes. The packet size j is set to be 378 bits – the closest to the maximum packet size of the reference application.

Table 2.1 shows the data partition for the three initially proposed coding schemes for code rate 1/2. Since the nearest code rate that the RS code can be set to 1/2 is 0.492 (31/63), the total number of encoded packets is three more than that of the other two coding schemes provided the user data to be transmitted should not be less than 32384 bits. Table 2.2 shows the data structure for code rate 1/3. This time as the RS code can have exactly 1/3 code rate, the number of encoded packets is the same for all three schemes.

Table 2.1: Data format for $Rc = 1/2$, for transmitting a set of user data of 32384 bits, j=378 bits.

Coding Schemes	**User data per packet** (bit)	**packet size,** j (bit)	**number of user packet,** k	**Total no. of transmitting packet**	**Actual amount of transmitting user data** (bit)	**Total transmitting budget** (bit)
Repetition	378	378	86	172	32508	65016
RS(63,31)	186	378	175	175	32550	66150
LT	378	378	86	172	32508	65016

2.3 Channel Models

Two channel models were employed in this project: the Binary Symmetric Channel (BSC) and the Gilbert-Elliott (GE) channel. This section introduces these two channel models and the way they were used in this project.

[3] Examples of allowable packet sizes of RS codes are shown in Table 4.1.

Table 2.2: Data format for $Rc = 1/3$, for transmitting a set of user data of 32384 bit, j=378 bits.

Codes	**User data per packet** (bit)	**packet size,** j (bit)	**number of user packet,** k	**Total no. of transmitting packet**	**Actual amount of transmitting user data** (bit)	**Total transmitting data** (bit)
Repetition	378	378	86	258	32508	97524
RS(63,21)	126	378	258	258	32508	97524
LT	378	378	86	258	32508	97524

2.3.1 The Binary Symmetric Channel

The Binary Symmetric Channel (BSC) represents a memoryless channel with a constant error rate as shown in Figure 2.1 where e represents a constant channel error probability. A bit level BSC was used for the RS code simulations. For the simulations of the repetition code and the LT code, the BSC is further extended to an erasure channel (BEC). The difference between a BSC and a BEC is only at the receiving end. The receiver keeps the unit error as it is after a BSC whereas after a BEC, the receiver erases the unit error leaving an unknown state in the slot where the unit error occurs. Another difference between a BSC and a BEC in the context of this project is that a BSC is applied to the RS code at the bit level and a BEC is applied to the repetition code and the LT code at the packet level. Figure 2.2 shows a BEC used in this project where E represents the packet erasure rate (PER) and $E = 1 - (1 - e)^j$ for transmission over a memoryless channel.

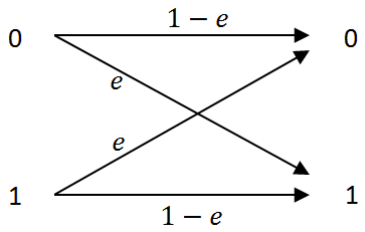

Figure 2.1: A BSC channel

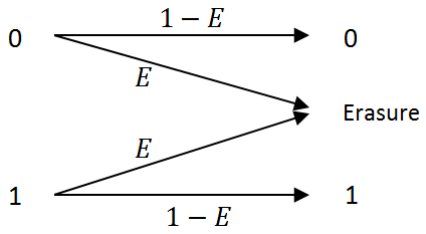

Figure 2.2: A BEC channel

2.3.2 The Gilbert-Elliott Channel

The Gilbert-Elliott (GE) channel is first introduced by Gilbert [14] and Elliott [11]. It is used as a statistical channel that models a burst error channel. It is a two-state Markov chain as shown in Figure 2.3, where each state is a memoryless channel. One state represents a good channel that has zero BER as defined by [14]; it is denoted as BER_g or PER_g which is set to zero. The other state represents a completely noisy channel [46]. In this thesis, the bad state is modified slightly. Instead of representing a completely noisy channel, the bad state can have a range of BER values. It is denoted as BER_b. It is converted to PER_b when the packet level BEC is used.

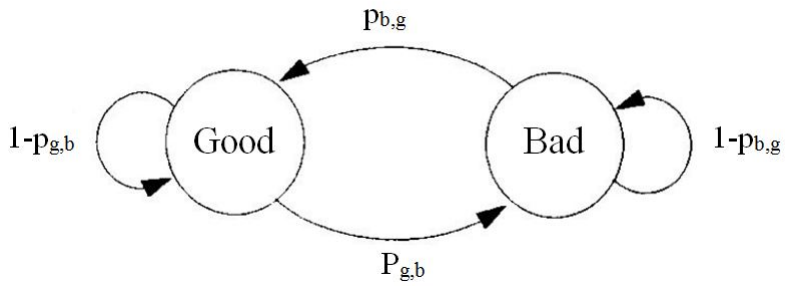

Figure 2.3: A Gilbert-Elliott channel transition diagram

Two other parameters for a GE channel are the transition probabilities: from the good to the bad state, $p_{g,b}$ and from the bad to the good state $p_{b,g}$. Their counterparts $p_{g,g} = 1 - p_{g,b}$ and $p_{b,b} = 1 - p_{b,g}$ are called the self transition probability; they are the probabilities that the channel stays in the same state in the current time slot as the previous time slot. The GE channel is used in such a way that the state can only change at the beginning of a transmitted packet and the BER (or PER) of the state affects the entire duration of the transmitted packet. Since the practical channel condition was unknown and channel measurement was not involved in this project, only arbitrary values are used for the parameters of the GE channel.

To keep the degree of freedom of the simulation manageable, the parameters are only set to certain selected values. The transition probabilities are set to be 0.1 ($p_{g,b} = p_{b,g}$ =0.1) to specifically represent a burst error channel. BER_g is set to be zero and the variable of the channel model is BER_b or PER_b. A range of BER_b is used to test the behaviour of the coding schemes. Figure 2.4 shows the burstiness of an example GE channel realisation when both $p_{g,b}$ and $p_{b,g}$ are set to 0.1. It shows 100 samples where one represents the channel in the bad state and zero represents the channel in the good state.

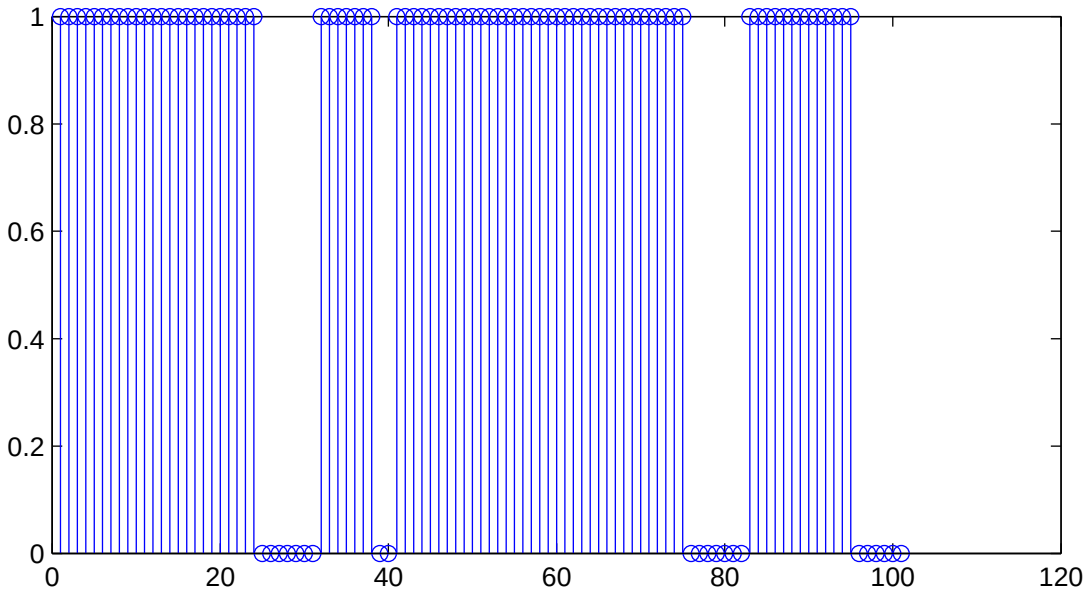

Figure 2.4: A bursty GE channel $p_{b,g} = p_{g,b} = 0.1$.

2.4 Performance Measures

There are two performance measures for comparing the candidate coding schemes: **success rate** and **time until (the receiver) switches off**. Success rate measures the average transmission success probability of the coding schemes and this shows the error correction capability of the coding schemes under a variety of channel conditions. The second measure, time until (receiver) switched off, measures the average number of packet transmitted before the receiver can switch off irrespec-

tive of reception success or failure[4]. This is a measure of time the receivers spent (and therefore the energy spent) on data reception with a coding scheme. This measure will be shortened as *transmission cost* in later context.

The encoding and decoding complexity of the candidate coding schemes is not examined. The only requirement in complexity is the candidate codes should not have more than polynomial complexity and this is ensured during the coding selection process.

2.5 Top-level Simulation Set-up

Both the results for success rate and transmission cost can be obtained simultaneously from the same experiment. The result for each simulation set-up is repeated until the confidence interval of the results falls to a pre-defined range. The method for calculating the confidence interval is based on [50, 51]. It is an estimation of the upper and lower bound over the range of where the true value could lie around the sample mean value with a specified confidence in percentage. It is applicable to a set of independent and identically distributed (i.i.d) random variables. In this case, the result from each individual experiment is an i.i.d random variable. The experiments are designed to be i.i.d by ensuring the random number sequences used for the experiments do not overlap.

The lower and upper boundary of the confidence interval is calculated by equation 2.1 and 2.2 respectively. $\hat{\mu}$ is the sample mean value and $t_{n-1,1-\frac{\alpha}{2}}$ is the upper $1-\frac{\alpha}{2}$ quantile of the student-T distribution with $n-1$ degree of freedom. α is the confidence level. $(1-\alpha)\cdot 100\%$ indicates the probability that the true mean lies within the confidence interval and α was set to 0.01.

$$l(n,\alpha) = \hat{\mu}(n) - t_{n-1,1-\frac{\alpha}{2}}\sqrt{\frac{S^2(n)}{n}} \tag{2.1}$$

$$u(n,\alpha) = \hat{\mu}(n) + t_{n-1,1-\frac{\alpha}{2}}\sqrt{\frac{S^2(n)}{n}} \tag{2.2}$$

$S^2(n)$ is the sample variance; it is given by

$$S^2(n) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu}(n))^2 \tag{2.3}$$

where n is the sample size and x_i is the result from an individual simulation.

[4] The receiver can switch off as soon as enough packets are collected for decoding or the total transmission budget is exhausted.

It was first decided to run the simulation enough times until the confidence interval fell to a pre-defined small value. However, the number of experiments needed was found to be unpredictable and hence it took a long time to run the experiments. In addition, the confidence interval had to be calculated after each new experiment which was inefficient. It was then decided to run the experiments a fixed (large) number of times and inspect the confidence interval of the set of results. If the confidence interval did not reach the pre-defined value, more simulation was executed. By running the simulation this way, the simulation time was found to be more controllable therefore this method was used for all subsequent simulations.

All results shown in this thesis have confidence intervals of less than 0.1% except for several figures in Chapter 5 that are indicated in Table C.3 in the Appendix. The relatively large confidence intervals for those figures are due to limited number of experiments executed[5] for large k values tested for the LT codes. Appendix C lists the details of the average confidence intervals for most figures in this thesis.

[5] due to long simulation time (more than two hours per experiment for $k = 10000$)

Chapter III

Repetition Codes

3.1 Background

Repetition codes are one of the simplest types of error correction codes. Messages can be repeated with different sequences to combat different channel loss patterns. Packet level repetition codes were used. The encoding process simply repeats the user data X times where $1/X$ is the code rate. At the receiver, a simple decoding process is involved. The checksum of each received packet is checked and the packet is dropped if the checksum is incorrect. Duplicate packets are suppressed. As soon as all user packets are received at the receiver, the data reception is successful and the receiver can be turned off[1]. If at least one user packet is missing at the end of the data transmission process, the data reception fails.

3.2 Simulation Models and Validation

This scheme is the baseline coding scheme - the coding scheme that is currently being used in the considered application. The encoder and decoder of this scheme is implemented in MATLAB and tested with the BEC and GE channels. The user data is sent X times where X can be 2 or 3. There were six transmission sequences tested and the sequences are specified in Table 3.1. Under the BEC, the success rate can be obtained with an analytical expression shown in Equation 3.1 due to the random and uniformly distributed loss characteristic and constant loss rate of the channel. The results that were produced by the simulations were compared with the analytical results to validate the simulation model. The model was then used to test the performance of the repetition codes under the GE channel for which no trivial analytical expression exists.

Analytical expression for the performance of repetition codes under the BECs:

$$\text{Success Rate} = (1 - (1 - (1 - BER)^j)^X)^k \tag{3.1}$$

[1] the receiving function of the receiver.

where BER stands for bit error rate, j is the user data packet length in bits, X is the number of repetitions or the reciprocal of the code rate R_c and k is the total number of user packets.

3.2.1 The Encoding and Decoding Process

The size of the user packets, j, and the total number of user packets k are initially set. The number of transmitted packets is $X \cdot k$. Each user packet has a sequence number. The sequence numbers are created according to the selected transmission sequence as shown in the second column in Table 3.1.

Packet erasures are then applied to the transmitted packets according to a set of binomial experiments. With a pre-defined BER of the channel a binomial experiment is performed on each packet with parameters j and p_{rep} where j is the packet size in bits, or the number of trials in the context of a binomial experiment, and p_{rep} is the probability of the success of a single bit ($p_{rep} = 1 - BER$) or the probability of success for each trial.

At the receiver, the transmitted packets that passed the binomial test are collected and their sequence numbers are stored. After all transmitted packets have gone through the binomial experiment, the success transmission of the user message is determined by inspecting the received sequence numbers. If there is at least one sequence number missing at the receiver, the transmission fails; otherwise the transmission is successful.

Table 3.1: List of sequence of data tested in repetition codes and examples.

Sequence Name	**Examples for** $X = 3$
Repeat Batch (REPB)	$1, 2, \ldots, k, 1, 2, \ldots, k, 1, 2, \ldots, k$
Repeat Packet (REPP)	$1, 1, 1, 2, 2, 2, \ldots, k, k, k$
Mixed Method 1 (MIX1)	$1, 1, 2, 2, \ldots, k, k, 1, 2, \ldots, k$
Mixed Method 2 (MIX2)	$1, 2, \ldots, k, 1, 1, 2, 2, \ldots, k, k$
Mixed Method 3 (MIX3)	$1, 2, \ldots, k, 1, k, 2, k-1, \ldots, k, 1$
Randomized (RAND)	$2, 1, k, 3, 4, 3, 3, k-1, 1, \ldots$

3.3 Performance Comparison between Different Repetition Code Sequences

Under the BEC, there is no difference in success rate performance among different sequences. This can be seen from Equation 3.1: the success rate does not depend on the transmission sequence.

It was validated by the simulation as shown in Figure 3.1. However, the transmission cost varies among different sequences as shown in Figure 3.2. The ordinate in Figure 3.2 shows the percentage of number of transmitted packets needed before the receiver switches off normalised by k, as a measure of the transmission cost. The abscissa is the PER. Among all the sequences, REPB and MIX3 have the smallest average reception time especially when the PER is small. In contrast, REPP can not have switch off early due to the nature of the repetition – each packet is repeated after its first occurrence and the earliest time the receiver can receive all user packet is after it has turned on for $X \cdot k - (X - 1)$ packet slots.

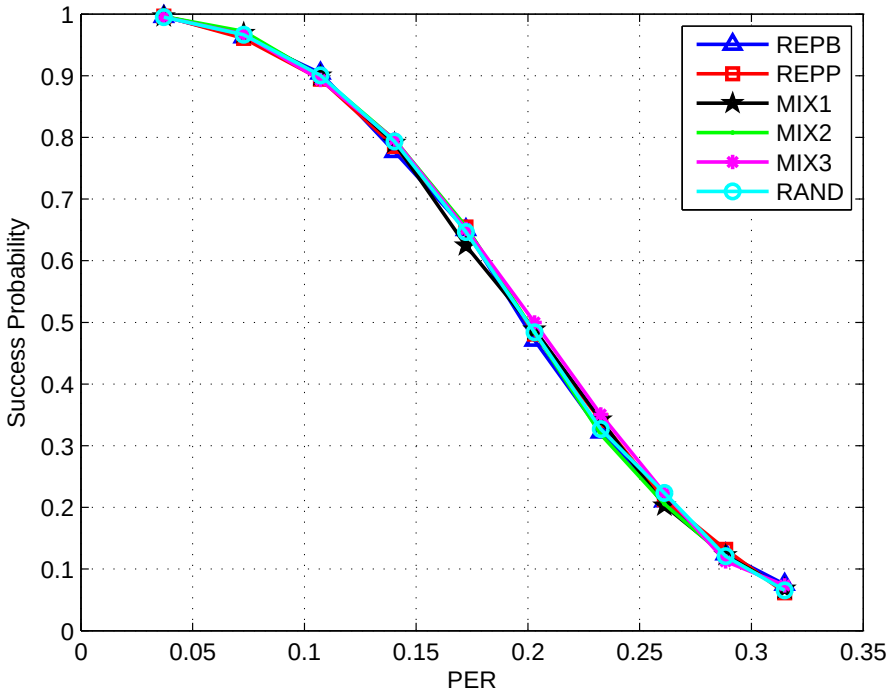

Figure 3.1: Success probability of repetition codes under BEC channel with different transmission sequences. $k = 86, j = 378, R_c = 1/3$.

Under the GE channels, different transmission sequences performs differently with different self transition probability values, as shown in Figure 3.3. The transition probabilities between good and bad states are assumed to be symmetrical. The abscissa indicates the self transition probabilities of the GE channel. Listed below are observations regarding to Figure 3.3 and the reasons for the behaviours of different sequences.

REPB and REPP are the two extreme cases in the arrangement of the transmitting packets. It

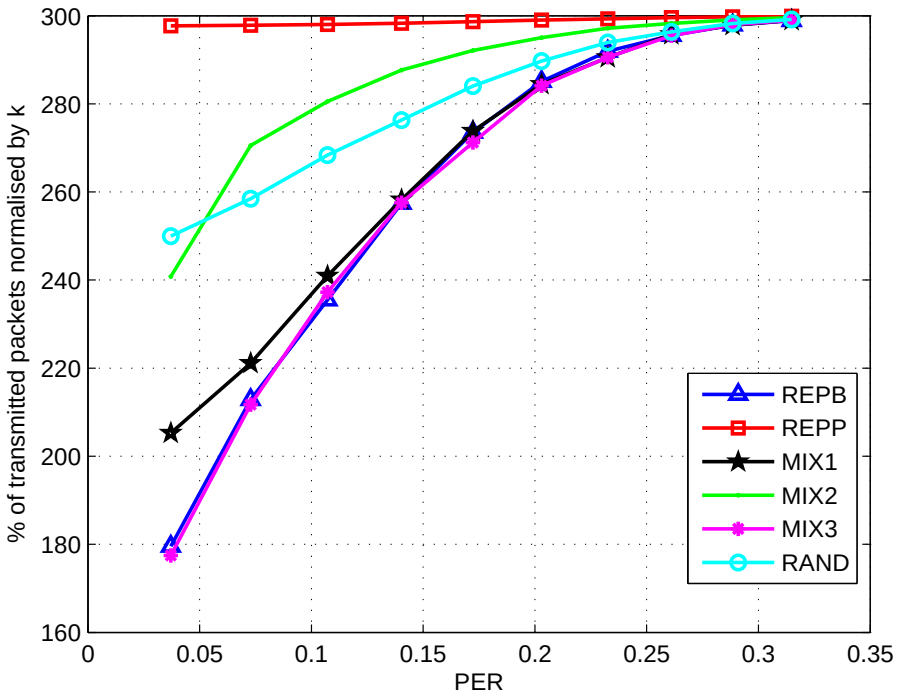

Figure 3.2: Transmission cost of repetition codes under BEC channel with different transmission sequences. $k = 86, j = 378, R_c = 1/3$.

can be seen from Table 3.1 that REPB has the maximum interval (k packets) between repeated packets and REPP has the minimum interval (1 packets). As shown in Figure 3.3, REPP varies the most between short and long burst error channels (represented by small and large self transition probability respectively). It has the best performance in short burst error channels and the worst performance in long burst error channels, whereas REPB barely has any variation between short and long burst error channels.

REPP performs much better than the rest of the sequences when the channel has short burst errors. It is because a packet is repeated immediately after its first transmission; there is a high probability that the lost packet can be recovered by its duplicates immediately before or after it because there is a high probability that the channel transits from one state to another within the time of the repetition of a single packet. In the case of long burst errors, there is a high probability that all duplicates of a particular packet are lost within a long continuous burst of errors, which leads to the permanent loss of a packet and causes the overall failure of the data transmission and therefore it is the worst sequence for long burst error channels.

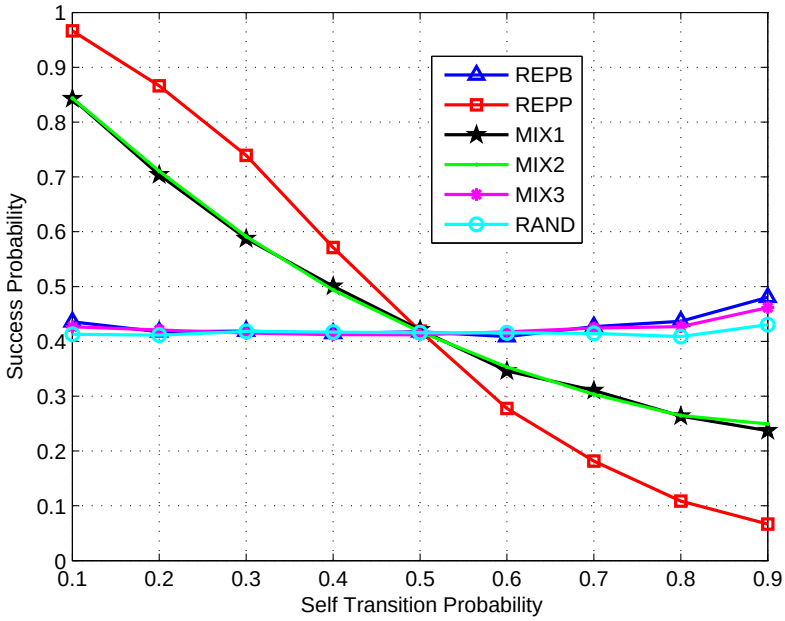

Figure 3.3: Success Probability of repetition codes under GE channel with different transmission sequences. $k = 86, j = 378, R_c = 1/3$,$PER_g$=0, PER_b=0.4.

REPB, MIX3 and RAND tend to have relatively constant success probabilities from short burst error channels to long burst error channels which indicates their capability to correct both random errors and burst errors is similar. MIX1 and MIX2 have similar performance and their lines lie between the performance of REPB and REPP because they contain both the features of REPB and REPP.

Figure 3.4 shows the transmission cost for all sequences under the GE channel for $R_c = 1/3$. All sequences have relatively constant reception time across all the self transition probability except MIX1. MIX1 sequence needs a significantly smaller transmission cost than the rest when the channel error burst is short. In the case of $R_c = 1/3$, this sequence repeats each packet twice first. The first two complete repetitions are sufficient for recovering all lost packet for most experiments under short burst error channel; this brings the transmission cost to a lower value.

As REPB has a relatively constant success rate across all state transition probabilities and it has good performance in transmission cost over the BEC, it was selected for the simulation to represent the performance of the repetition code.

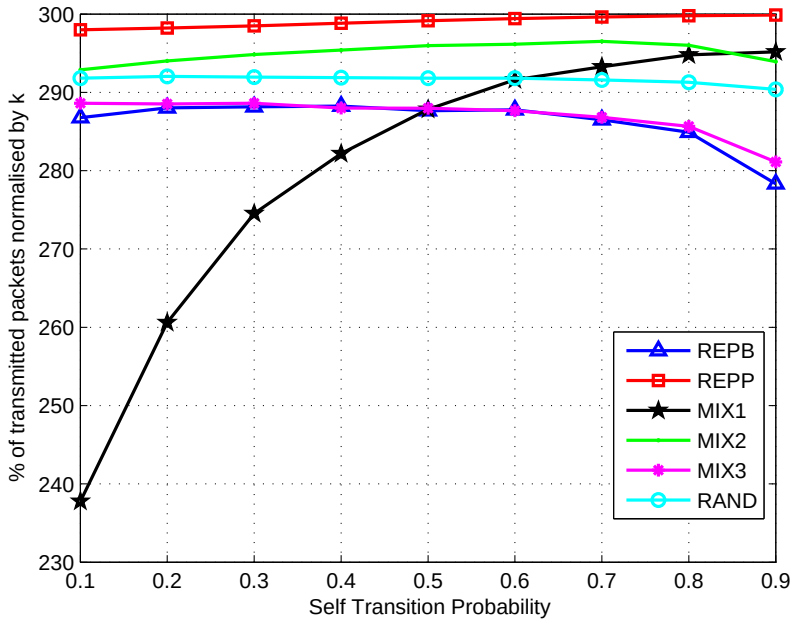

Figure 3.4: Transmission cost of repetition codes under GE channel with different transmission sequences. $k = 86, j = 378, R_c = 1/3$, PER_g=0, PER_b=0.4.

Chapter IV

Reed-Solomon (RS) Codes

4.1 Background

Reed Solomon (RS) codes were first introduced by Irving Reed and Gus Solomon in a paper called "Polynomial Codes over Certain Finite Fields" [36] in 1960. RS codes have been widely used in digital communications and data storage systems. Applications that use RS codes include satellite and space communications, digital HDTV and CD players [21, 33]. RS codes are a type of non-binary cyclic codes and they can be seen as a special case of non-binary BCH (short for Bose-Chaudhuri-Hocquenghem) codes. RS codes are Maximum Distance Separable (MDS) codes which means the minimum distance between codewords is $n_{RS} - k_{RS} + 1$ which is the highest possible distance[1] [29]. This indicate that RS codes have a very strong error correcting capability.

RS codes can be used as error correcting codes as well as erasure codes. In this thesis, RS codes were used as systematic intra-packet error correcting codes. Redundant error-correcting data are appended to each block of user message and the actual user message in each block is not altered. An RS code can be written in the form RS(n_{RS},k_{RS}) where n_{RS} is the total number of symbols per encoded packet and k_{RS} is the number of user message symbols per encoded packet[2]. An RS encoded packet can correct up to t symbol errors where $t = (n_{RS} - k_{RS})/2$. Figure 4.1 shows the structure of a systematic RS encoded packet. The fact that an RS code corrects errors at symbol level makes it particularly good at correcting burst errors[49, 43].

The construction of an RS code uses a finite field, sometimes referred to as the Galois field (GF). The most popular RS codes operate in GF(2^m) [33] and this thesis also only focuses on RS codes operate in GF(2^m). The encoded packet size of the RS codes that build on GF(2^m) should match the symbol size in a way that if a symbol contains m bits, the packet size must be $2^m - 1$

[1] n_{RS} and k_{RS} denote the total number of symbols per encoded packet and number of user data symbols per encoded packet respectively.

[2] The notations n_{RS} and k_{RS} are used in order to distinguish them from n and k which mean the total number of encoded packets and total number of user packets respectively in the context of this thesis.

symbols or $(2^m - 1) \times m$ bits, assuming no puncturing or shortening technique is used[3]. With this assumption being true, there can only be an odd number of user message symbols contained in a packet because an RS encoded packet always contains an odd number of symbols and the number of parity symbols must be even: two parity symbols correct one symbol error – one symbol finds the location and the other one corrects the error. Examples of allowable packet sizes for RS encoded packets are shown in Table 4.1.

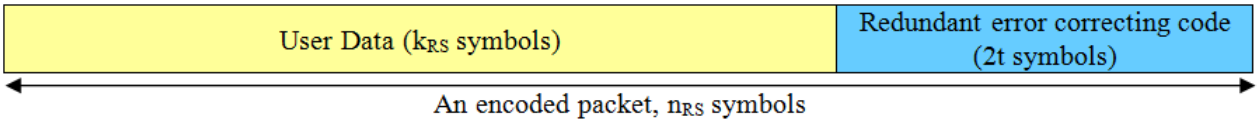

Figure 4.1: Structure of an RS encoded packet.

Table 4.1: Allowable RS encoded packet sizes for $m \in \{4, 5, 6, 7, 8\}$, where code construction uses GF(2^m).

symbol size, m	**symbols per encoded packet,** n_{RS}	**bits per encoded packet**
4	15	60
5	31	155
6	63	378
7	127	889
8	255	2040

4.2 Simulation Model and Validation

The available user data is split into packets. The symbol size that was determined to be used was $m = 6$ which gives a packet size of 63 symbols or 378 bits. This was chosen because it is the closest allowable RS code packet size to the maximum packet size of the reference application, which is 62 bytes or 496 bits.

The simulation for the RS coding scheme uses the RS code's encoder and decoder from the MATLAB's communications toolbox. The Berlekamp-Massey decoding algorithm is used for decoding [32, 48]. The RS encoder and decoder objects are created by MATLAB functions `comm.`

[3] The RS code length can be adjusted by shortening or puncturing, however this is not explored in this thesis.

`RSEnocder` and `comm.RSDecoder`. The user data is generated and fed into the RS encoder. The user data has k_{RS} randomly generated integers betweens 0 and n_{RS} where $n_{RS} = 63$ in this case; each integer represents a symbol.

Once the encoded packets are constructed, the chosen channel model with the chosen BER is applied. The errors are initially generated as binary numbers according to the BER. They are then converted to m-ary integers (where m=6 in this case and this is the symbol size) and added to the encoded packets.

The resultant noisy RS encoded packet is then decoded by the MATLAB RS decoder. By comparing the decoded user data with the original generated user data, the program decides whether the transmission is successful. Since this scheme is used as intra-packet coding scheme, packet erasures are not applicable. Instead, a bit-level BSC model is used for this scheme for both memoryless channel and burst error channel models.

The RS code's encoder and decoder are built according to the RS code error detection and correction example in [31]. The error performance produced by the RS simulation model built in this project is compared with the approximation made by the MATLAB function `bercoding(EbNo, 'RS','hard',n,k)`. Documentation of this function can be found in [30]. As shown in Figure 4.2, two curves have very similar performance which validates the RS simulation model built.

4.3 Performance of RS Code with Different Block Length

In order to make the number of simulations feasible, only one packet size is used for comparing the performance among all coding schemes. However, the effect of different packet (block) sizes on the performance of RS code is examined in this section.

Again, the total message size is 32384 bits. The block lengths tested are $n_{RS} = 15$, 31, 63, 127 and 255. The code rates tested are 1/2 and 1/3. Table 4.2 shows a list of RS code packet sizes tested for each code rate, and number of encoded packets needed for transmitting 32384 bits of user data. The packet numbers were rounded up. Figures 4.3 and 4.4 show the success rate of each packet size under the BSC for code rate 1/2 and 1/3 respectively. Figure 4.5 shows a bar graph of the maximum BERs that each RS codes can reach before the success rates drop below 99%.

From Figure 4.5 it can be seen that as the block size increases, a close to 100% success rate can be maintained at higher BERs. For $R_c = 1/2$, the difference between RS(15,7) and RS(255,127) is 0.018. For $R_c = 1/3$, the difference is 0.024. The biggest gap is between $n_{RS} = 31$ and $n_{RS} = 63$. The trade-off for higher channel error tolerance is the number of `XOR` operations needed for encoding and decoding. Whenever the symbol size is increased by one, the total number

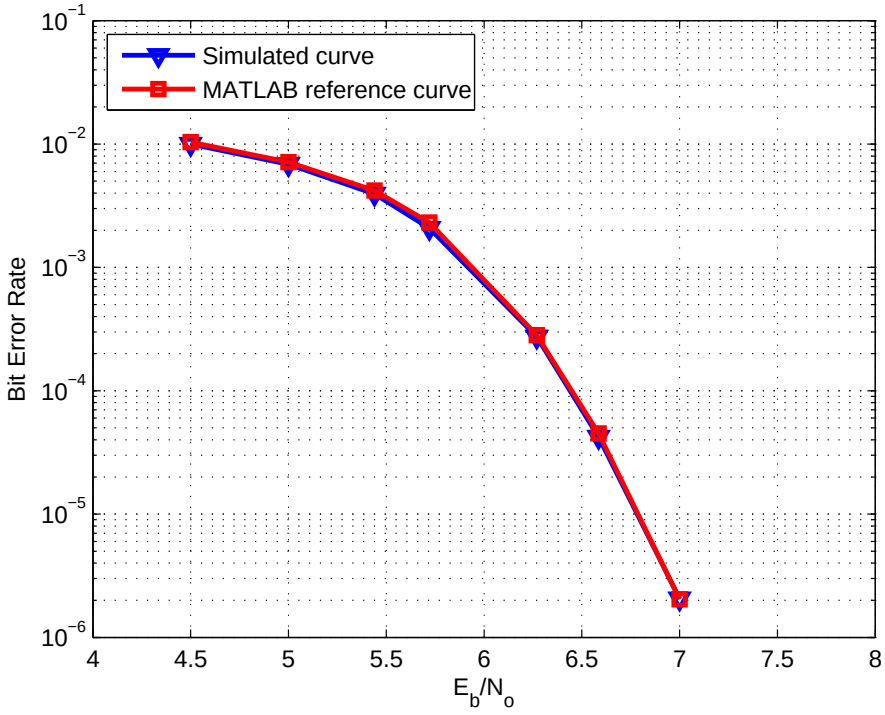

Figure 4.2: Error performance comparison between the MATLAB built-in approximation and the simulation model built in this project for RS(255,239) code.

of XOR operations needed for encoding and decoding is doubled[4]. The calculation is based on [5]; the number of XOR operations needed is similar for encoding and decoding, which is $k_{RS} \cdot (n_{RS} - k_{RS}) \cdot m \cdot k/2$.

The results shown in Figure 4.5 may be affected by small differences in code rates, see table 4.2, among different block lengths. For $R_c = 1/2$, the code rate decreases as the block length decreases so this makes the performance comparison among difference block length conservative. For $R_c = 1/3$, the code rates of three of the block lengths are consistent which gives a consistent comparison among the majority of the block lengths.

[4] The finite field arithmetic operations for RS encoding and decoding can be replaced by XOR operations. Details see [3].

Table 4.2: RS code packet sizes tested for each code rate, and number of encoded packets needed for transmitting 32384 bits of user data.

Code Rate	RS code	Actual Code Rate	Total Encoded Packets Needed
$1/2$	RS(15,7)	0.467	1157
	RS(31,15)	0.484	432
	RS(63,31)	0.492	175
	RS(127,63)	0.496	74
	RS(255,127)	0.498	32
$1/3$	RS(15,5)	1/3	1620
	RS(31,11)	0.355	589
	RS(63,21)	1/3	258
	RS(127,43)	0.370	108
	RS(255,85)	1/3	48

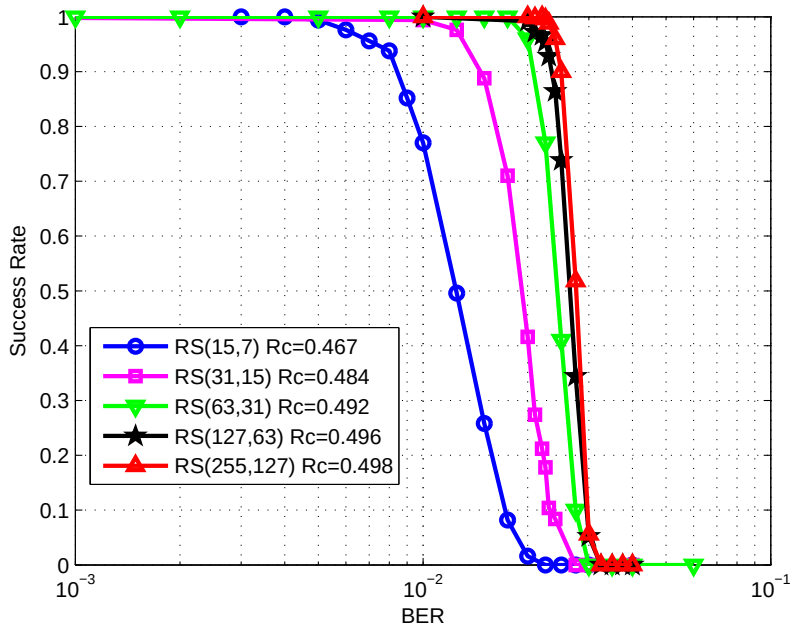

Figure 4.3: Success rates of different RS encoded packet sizes, $R_c \approx 1/2$, total user data = 32384 bits.

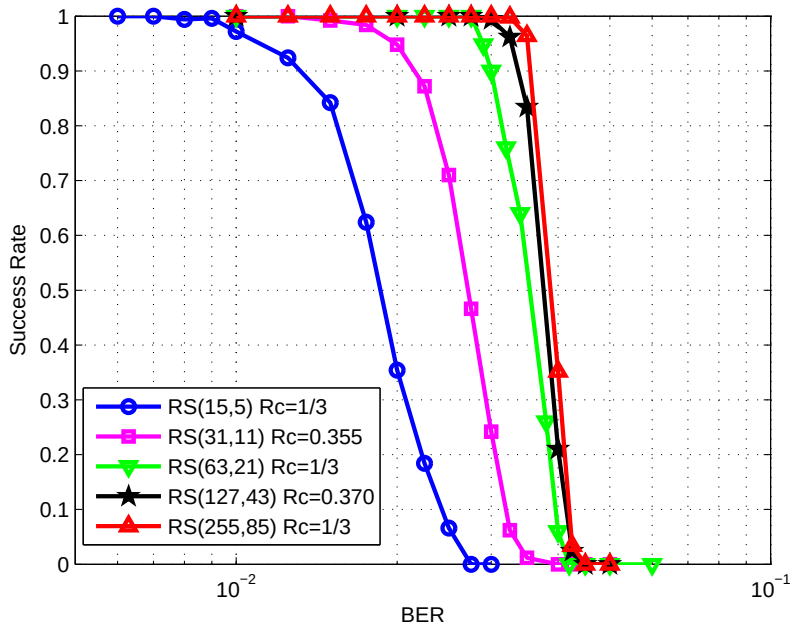

Figure 4.4: Success rates of different RS encoded packet sizes, $R_c \approx 1/3$, total user data = 32384 bits.

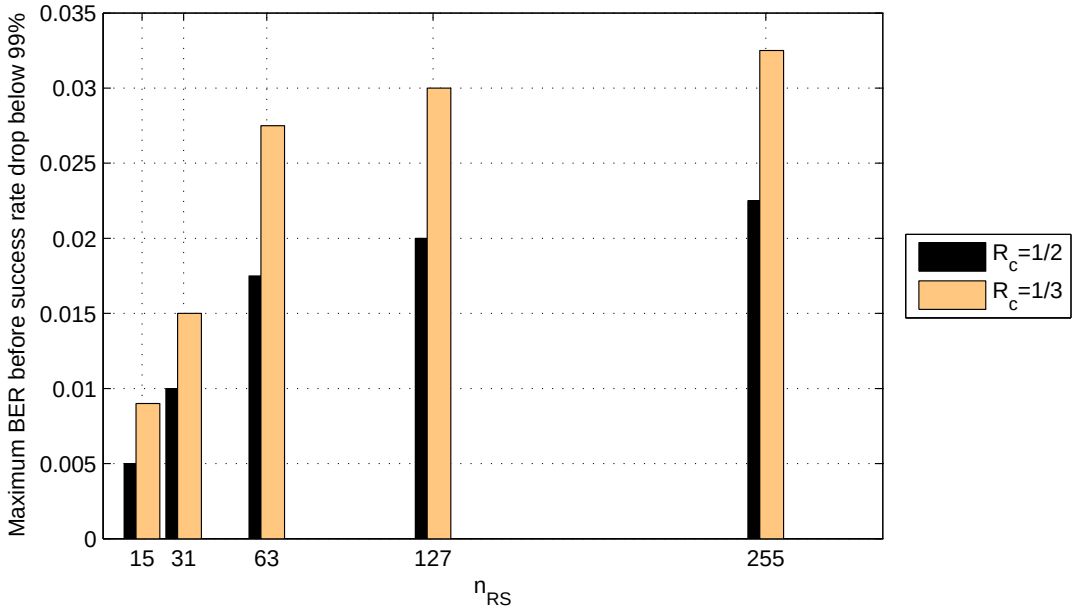

Figure 4.5: The maximum BERs for the RS codes tested before the success rates drop below 99%, total user data = 32384 bits.

4.4 An Observation on the Code Rate of RS Codes

This section analyses RS codes from the perspective of physical layer error performance. This is a piece of extra work done which is not on the main line of the work and does not have direct relationship with the results generated later. In this section, the analysis is done on the basis that the energy per user symbol is assumed to be fixed. However it is not applicable to the main work of this thesis, where the energy per channel symbol is assumed to be fixed.

Figure 4.6 shows the error performance of RS codes for n_{RS}=63 where k_{RS} ranges from 59 to 11. The figure shows that the error performance does not necessarily become better as the code rate goes lower provided that the energy used for each user symbol is fixed. The best error performance for the range of k_{RS} tested is achieved by RS(63,41) for E_b/N_0 greater than about 4.2dB. A larger k_{RS} or smaller k_{RS} has a worse performance. This is because the error performance is affected by two things: coding and modulation. As more coding information is added, the coding part improves the performance, it also means less energy per channel symbol which makes the demodulation process more likely to make an error. As shown in Figure 4.6, as the code rate decreases from $k_{RS} = 59$ to $k_{RS} = 41$, the effect of coding dominates; this gives better error performance as the code rate decreases. As the code rate decreases further, the modulation effect wins and the error performance start to degrade. This phenomenon is also described in [43].

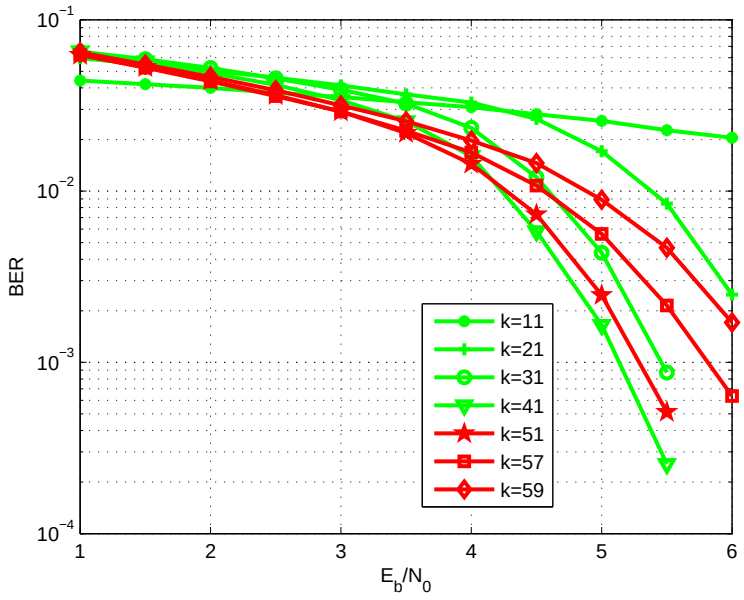

Figure 4.6: Error performance of RS codes $n_{RS} = 63$. BPSK modulation, AWGN channels.

Chapter V

Luby-Transform (LT) Codes

This chapter contains the most important work of this project: the investigation of a subclass of fountain codes called LT codes. A novel design of an improved version of the LT codes is also introduced in this chapter. This chapter is split into five main sections. Section 5.1 provides background information on LT codes – the initial design and the basic operations. Section 5.2 describes the MATLAB implementation of LT codes and the validation of the implementation. Section 5.3 introduces the novel contribution – the design of the LTAM codes. Section 5.4 compares the performance of the LTAM codes with that of the original LT codes as well as improvements reported in other literature. Finally, Section 5.5 concludes the chapter.

5.1 Background

A key technical article that introduces the idea of the fountain codes is [5]. This paper explains the digital fountain concept, its advantages for disseminating large sized data in broadcast/multicast systems, and the initial attempts in realising the idea. The paper discusses the impracticality of using erasure RS codes due to the limitations on the packet size and the exceedingly high encoding and decoding times for large data block sizes. The first approximation of the digital fountain was Tornado codes followed by a more practical realisation: LT codes. The paper then demonstrates the significant encoding and decoding efficiency improvement made by the two realisations over the RS codes. Knowing the advantages of the fountain codes for multicast systems, it was decided that the performance of LT codes be tested.

5.1.1 The Concept of Tornado Codes

Descriptions and explanations of Tornado codes can be found in [27, 26, 25, 5]. It was the code developed to realise the idea of the digital fountain after finding problems in using erasure RS codes. In contrast with RS codes' more than quadratic encoding and decoding time for practical values of n [5, 3, 41], Tornado codes achieve linear encoding and decoding times with the cost of

small additional overhead at the receivers [26, 18]. Tornado codes are systematic codes and they are "rateful" – the code rate must be predetermined.

The encoding principle of the Tornado codes is: n encoded packets are formed by k user packets and l redundant packets. The redundant packets, corresponding to check nodes, are formed by linear combinations of either a randomly chosen subset of the k user packets (these forms layer one of the check nodes), or a randomly chosen subset of the already constructed check nodes (these form the subsequent layers of the check nodes). The encoding process can be represented by a cascade of sparse bipartite graphs as shown in Figure 5.1. The number of packets that are included in a check node (called the degree) follows a predetermined degree distribution. The check nodes are appended after the message nodes. The last layer can be encoded with a conventional code for better protection against channel errors.

The challenge in designing efficient Tornado codes is the excessively complex method for generating and distributing the degree information. Although a linear programming tool described in [26, 25] can be used for finding good degree distributions, the encoding and decoding processes are still cumbersome due to the fact that the degree distribution needs to be recomputed for different user data lengths and code rates, and all current packet mapping information needs to be either constructed or transmitted explicitly to the receivers for decoding [23]. Another disadvantage of Tornado codes is that they are not entirely suitable for channels with high loss rates because the practical code rates for Tornado codes are small (e.g. 1/4) [5], and the overhead also becomes larger for small k as suggested in [5].

Tornado codes are not included in this study due to their disadvantages mentioned above and most importantly because LT codes are capable of replacing the Tornado codes. LT codes inherit the key feature of the Tornado codes: constructing encoded packets by linear combining a set of randomly chosen user packets. LT codes have even more advantages such as a much simpler structure and the rateless feature.

5.1.2 Concepts of LT Codes

LT codes are the first practical realization of the digital fountain concept [24]. The term fountain describes the data dissemination process. There can be infinite number of encoded packets produced at the transmitter which is like a water fountain which has an endless supply of water drops (the water drops are analogous to the encoded packets). The receivers are like cups that collect water drops from the water fountain. As soon as enough water drops are collected to quench one's thirst (enough packets are collected for decoding), the cup can be moved away from the fountain

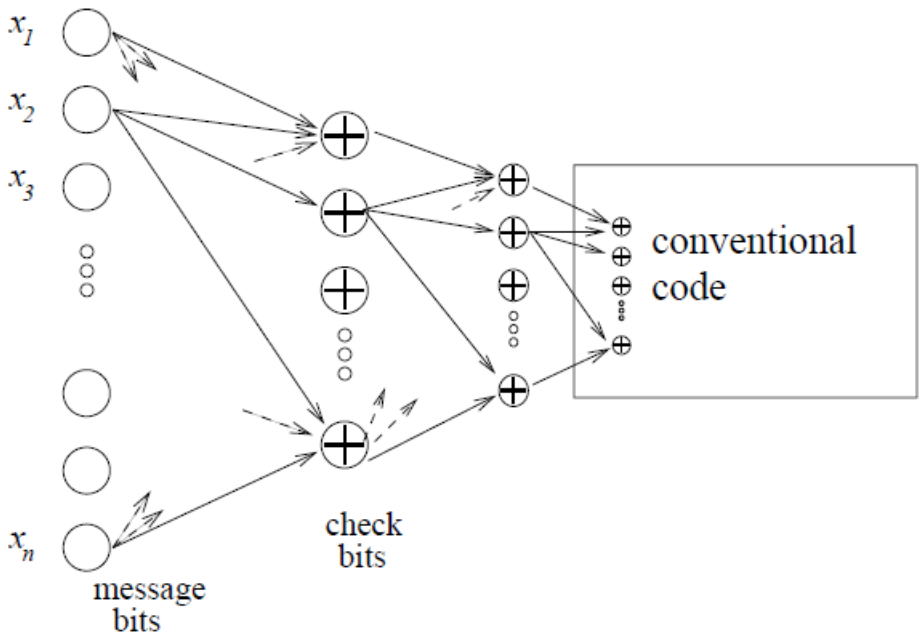

Figure 5.1: The encoding process of the Tornado code. Picture modified from [26].

(the reception process can be finished).

LT codes are designed for efficient and reliable dissemination of finite length large-sized data blocks for a large number of heterogeneous receivers[1] for broadcast or multicast transmissions[5]. LT codes are erasure codes; erroneous packets[2] are erased at the receiver. LT codes offer no mechanism for fixing erroneous packets; only packets that are received correctly can be utilised by the decoding process. The correctness of a packet can be determined in various ways e.g. the checksum of the packet or another intra-packet error correcting or detecting coding scheme such as RS codes. This also has an impact on the performance of the codes, and this is discussed in Section 6.3.

As a subclass of fountain codes, LT codes inherit the properties of fountain codes:

1. **"Rateless"** No code rate needs to be determined *a priori*; there can be an infinite number of encoded packets constructed. On average, the receiver can decode the data by successfully receiving any $(1 + \epsilon) \cdot k$ number of encoded packets [23] where k is the number of user

[1] Heterogeneous receivers: receivers with varying channels which do not have to be synchronised.

[2] The term "packet" in this case can be considered the same as "symbols" – the basic building blocks of LT codes data structure.

packets and ϵ is the relative overhead that is dependent on k and independent of the packet size.

2. **Independent data reception** The data reception processes for different receivers are entirely independent from one another. They allow receivers with good channels to switch off early while they can supply sufficient data for receivers with bad channel conditions for successful reception. Therefore, LT codes are a good data distribution method for disperse or heterogeneous receivers.

In the next three sub-sections, the encoding and decoding processes of the original[3] LT codes, as well as the degree distribution used for constructing LT encoded packets are introduced.

5.1.3 Encoding Process

The user data is initially divided into packets of equal size; they are called the user packets. An LT encoded packet is constructed by linearly combining (i.e. `XOR`'ing) a subset of the user packets, plus some additional information for decoding. The number of input user packets used to generate an encoded packet is called the degree of the encoded packet. The degree is randomly selected between 1 and k from the Soliton distribution that is described in Section 5.1.5. The actual user packet(s) mapped to an encoded packet are selected randomly from a uniform distribution; they are called the neighbours of the encoded packet. The encoding process of LT codes can be represented by a bipartite graph as shown in Figure 5.2.

In Figure 5.2, there are four user packets and five encoded packets constructed and they have degrees 2,3,2,1 and 2 respectively. The first encoded packet linearly combines the first and third user packet. The second encoded packet linearly combines of the first three user packets and so on. The term edge in Figure 5.2 represents a link between an encoded packet to one of its neighbours and this term becomes useful when explaining the LT decoding process.

5.1.4 Decoding Process

Before introducing the decoding process, it is necessary to make distinctions between three terms that are used in explaining the LT decoding process: encoded packets, received packets and user packets. **Encoded packets** refer to the LT encoded packets constructed at the transmitter. **Received**

[3] Much work has been done on LT codes in maximizing efficiency. In order to distinguish the original design of LT codes from the additional work done by other authors and also in this thesis, the LT codes described in [23] will be called the original LT codes in subsequent sections.

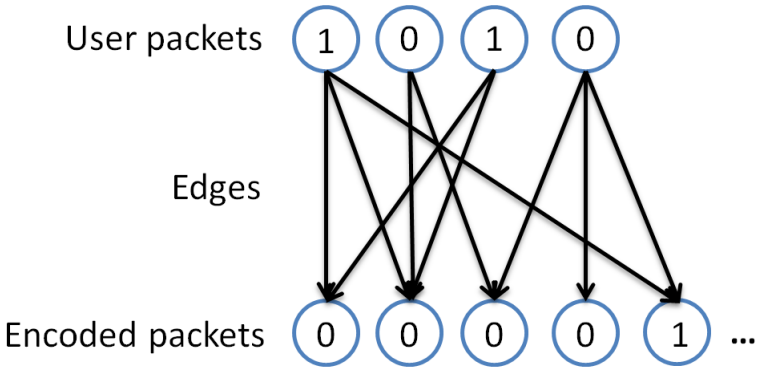

Figure 5.2: LT encoding with $k = 4$ shown as bipartite graph for the case where each user packet is a single bit.

packets refer to the received LT encoded packets at the receiver. Under no-loss channels, received packets are exactly the same as the encoded packets whereas under lossy channels, received packets are only the portion of the encoded packets that have successfully reached the receiver. In another word, received packets are encoded packets but the reverse is not necessarily true. **User packets** as mentioned in Section 5.1.3 refer to packets that are being encoded at the transmitter and decoded at the receiver. The encoded packets are assumed to have the same size as the user packets. In real systems, there will be overhead added to the encoded packets such as checksum, sequence numbers and other control information. However, since this is not relevant for the purpose of this project, the overhead is ignored in the simulation.

Two pieces of information need to be known for decoding the received packets: the degree of each encoded packet and the neighbours of each encoded packet. This information can be communicated in two ways:

1. the degree and mappings of the encoded packets can be transmitted explicitly from the encoder to the decoder[23].

2. the transmitter and receivers can apply the same random generator in which case only a seed for each encoded packet needs to be explicitly conveyed to the receivers [24].

By knowing the neighbours of each received packet, a decoding matrix can be formed at the receiver. A decoding matrix is an $n' \times k$ binary sparse matrix that represents the relationship between the encoded packets and the user packets where k is a fixed value that represents the number of existing user packets, and n' represents the number of received packet where $n' \leq n$

and can be increased according to the number of packets that have been received at the receiver[4]. Each entry of the matrix indicates whether an edge exists between the received packet i and the user packet j, where i and j are the row index and column index of the decoding matrix respectively. The decoding process of LT codes can be seen as solving a set of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the decoding matrix with dimension $n' \times k$, $\mathbf{x}$ represents the user packets that are desired to be discovered at the receivers and $\mathbf{b}$ are the received packets. The next paragraph describes the decoding method introduced by Luby, the inventor of LT codes. However, the decoding method used in this project is slightly different and it is introduced in Section 5.2.2.

According to [23], the decoding process consists of two stages: recover and process. The decoding process starts with the recover stage – decoding of a received packet that has only one neighbour (degree 1 encoded packet). Once a user packet is decoded, it is put into the ripple[5]. The user packets in the ripple then undergo the process stage – each user packet in the ripple is subtracted from all the received packets that contain it, i.e. all the received packet(s) that have an edge connecting to them. Once the user packets in the ripple are processed, they are removed from the ripple. Then, the recover and process steps start over again – the encoded packets that have exactly one neighbour left are decoded and the decoded user packets stay in the ripple until they are processed.

Take the code from Figure 5.2 for example. If all encoded packets are received, which makes them all received packets, user packet four can be decoded with the degree-1 received packet four as shown in Figure 5.3. This is the recover step. The dotted edge shown in Figure 5.3 is then removed and user packet four enters the ripple. In the process step, the value of user packet four is subtracted from every received packet that has an edge connecting to it (encoded packets three and five in this case) and the edges are removed as shown as the dotted line in Figure 5.4. At this stage, user packet four can be removed from the ripple because it has been processed and no edge is connected to it. The next iteration of decoding then can be started with the recover step – decoding of what is now the degree-1 received packets three and five as shown in Figure 5.5 and this leads to the recovery of user packets one and two.

---

[4] n is the total number of encoded packets. A full list of symbols is presented at the beginning of this thesis.

[5] **Concept of Ripple:** The term ripple is introduced in [23] and is an important concept for designing good degree distributions. The ripple acts like a container which contains the newly discovered user packets that are about to undergo the second decoding stage: process (the process stage will be introduced immediately after this footnote). Once a user packet is processed, it is removed from the ripple. Whenever a new user packet is discovered, it is entered into the ripple. The decoding process terminates when the ripple becomes empty. If the user packets are all discovered at that stage, the decoding process is successful; otherwise, the decoding fails if no more received packets enter the ripple.

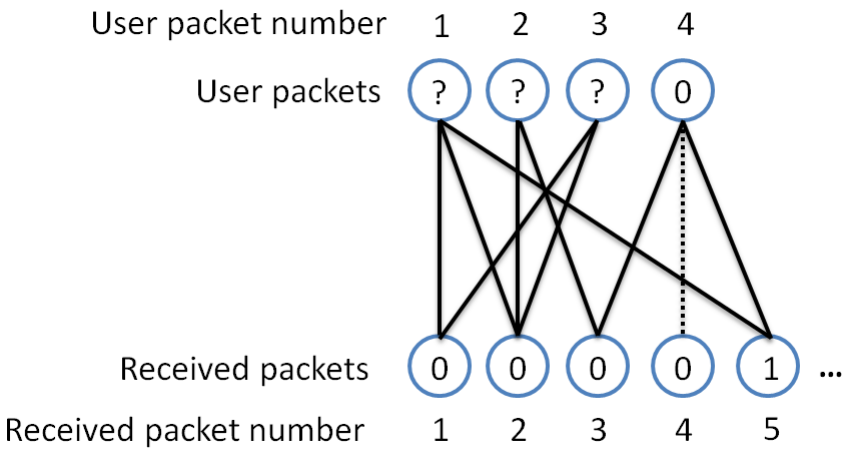

Figure 5.3: Recover step: degree-1 received packet four decodes user packet four.

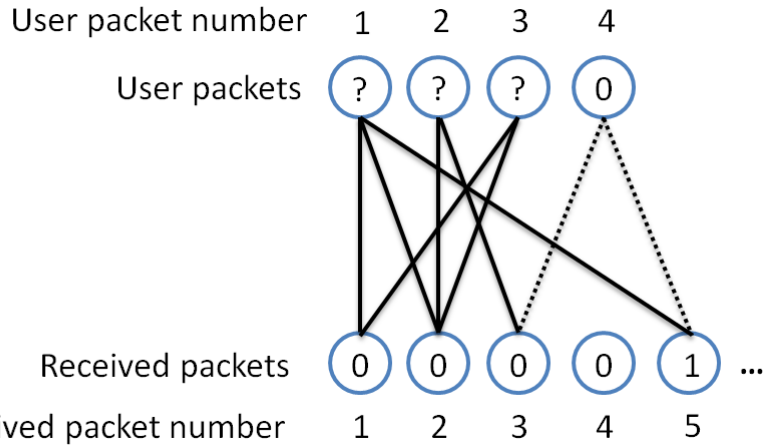

Figure 5.4: Process step: Removal of the edges of user packet four that is in the ripple.

This decoding process is repeated until the ripple becomes empty and no more user packets enter the ripple. If all k user packets are discovered at the end of the process, the decoding succeeds. Otherwise, more encoded packet(s) need to be received for decoding the remaining unknown user packet(s).

After further observation and analysis of the decoding process, it was found that there is another situation which has never been explicitly mentioned in any literature, that can initiate the decoding process (other than degree 1 encoded packets). This could potentially speed up the decoding process. This decoding method is used in the simulations and it is described in Section 5.2.2. The performance between the two decoding method has not been compared mainly due to the limited

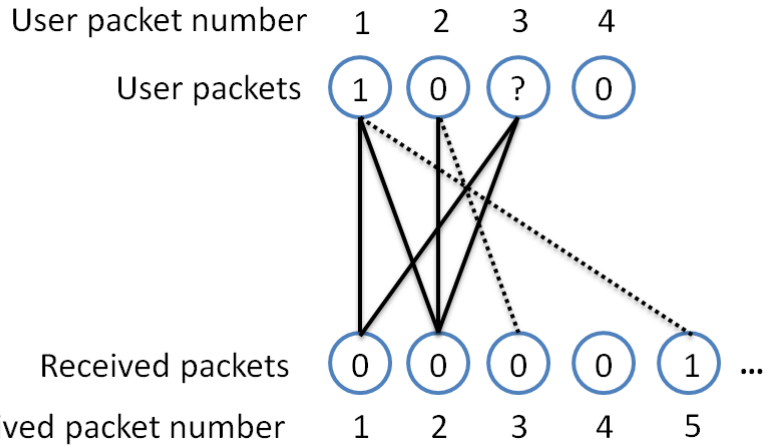

Figure 5.5: Recover step in the second decoding iteration: decoding of now degree-1 received packets three and five.

time frame. This is mentioned in Section 7.3 as one of the future works.

5.1.5 Soliton Degree Distribution

Two classes of Soliton distributions were designed for the degree distribution of the original LT codes in [23]: the Ideal Soliton distribution and the Robust Soliton distribution. These two degree distributions are described in the next two subsections.

Ideal Soliton Distribution

The Ideal Soliton distribution is given by the following probability mass function:

$$\begin{aligned} \rho(1) &= \frac{1}{k} \\ \rho(d) &= \frac{1}{d(d-1)} \quad \text{for } d = 2, 3, \ldots, k \end{aligned}$$

where k is the number of user packets and $\rho(d)$ is the probability that an encoded packet has degree d. Figure 5.6 shows the probability mass function of the Ideal Soliton distribution for k=100.

The Ideal Soliton distribution is observed to work poorly in practice [23, 8] because the size of the ripple is very small (expected size is one). Variation in the ripple size is very likely to make the ripple disappear which causes decoding failure. Although the Ideal Soliton distribution is almost never used in practice, it serves as a starting point for building the Robust Soliton distribution,

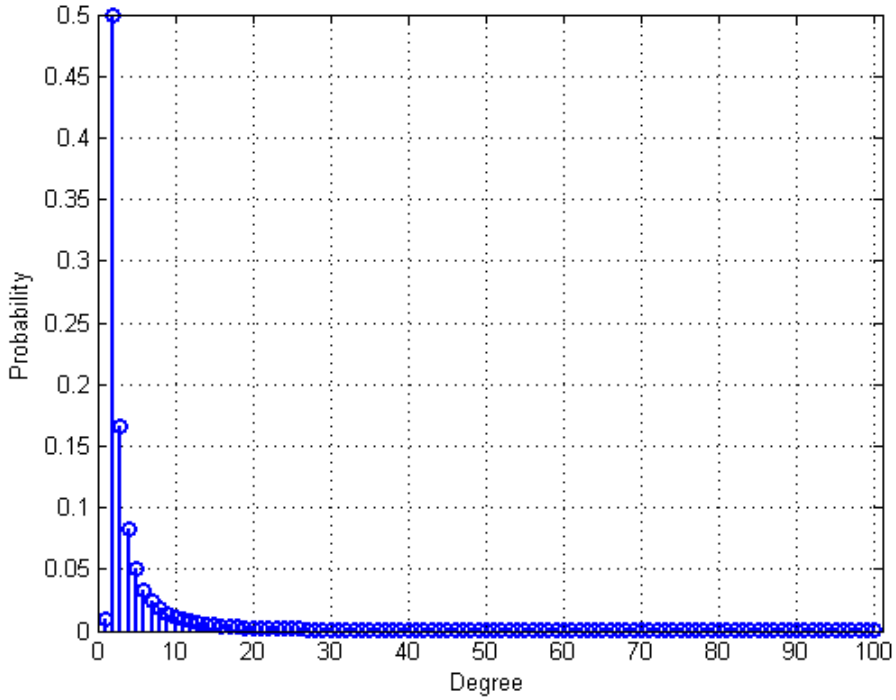

Figure 5.6: Probability mass function of the Ideal Soliton distribution for k=100.

which gives much better performance in terms of preventing premature termination of the decoding process and keeping the overhead needed for decoding small.

Robust Soliton Distribution

The Robust Soliton distribution improves the Ideal Soliton distribution in two respects: the expected ripple size is increased so that the ripple will not disappear with high probability [23], and it also ensures that the ripple does not grow too large, so as to avoid excessive redundancy in the decoding process. Two parameters were introduced: c and δ, where c is a free constant parameter that takes values between 0 and 1, and δ is a bound on the decoding failure probability after a certain number of encoded packets are received [23, 8]. The Robust Soliton distribution is given by:

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta} \quad \text{for } i = 1, \ldots, k$$

where $\rho(\cdot)$ is the Ideal Solition distribution,

$$\tau(i) = \begin{cases} R/(ik) & \text{for } i = 1, \ldots, k/R - 1 \\ R \cdot \ln(\frac{R}{\delta})/k & \text{for } i = k/R \\ 0 & \text{for } i = k/R + 1, \ldots, k \end{cases}$$

where $R = c \cdot \ln\left(\frac{k}{\delta}\right) \sqrt{k}$, and the normalization factor β is

$$\beta = \sum_{i=1}^{k} \rho(i) + \tau(i)$$

Figure 5.7 shows the probability mass function of the Robust Soliton distribution for $k = 100$ with $\delta = 0.5, c = 0.03$. It can be observed in Figure 5.7 that a spike occurs at degree 63. Compared with the Ideal Soliton distribution, the Robust Soliton distribution enables the encoding process to draw a higher degree, such as 63 in this case, with a higher probability to ensure enough user packets stay in the ripple and therefore decreases the probability of early disappearance of the ripple which causes decoding failure.

5.2 Simulation Model and Validation

Since there is no existing fountain code implementation found in the MATLAB toolboxes, building an encoder and decoder for the LT codes was undertaken. In this section, some important implementation features of LT codes are introduced and the validation of the encoder and decoder is presented.

5.2.1 MATLAB Implementation of the LT Encoding and Channel Erasure Process

At the encoder, the degree of each encoded packet is drawn independently from the Robust Soliton distribution. The parameters δ and c are chosen to be 0.5 and 0.03 respectively due to the relatively small overhead produced by these two values as shown in Figure 5.8.

For the transmissions of a message with k packets, k encoded packets are firstly constructed by the encoding process mentioned in Section 5.1.3. Packet loss is then applied to the encoded packets according to the constant packet erasure rate (PER) of the channel if the BEC is used or the current packet erasure rate if the GE channel is used. The successful reception of an encoded packet is determined by a random experiment with a threshold of P_{suc} where $P_{suc} = 1 - PER$ or $(1 - BER)^j$. An encoded packet is lost if the randomly generated probability exceeds the

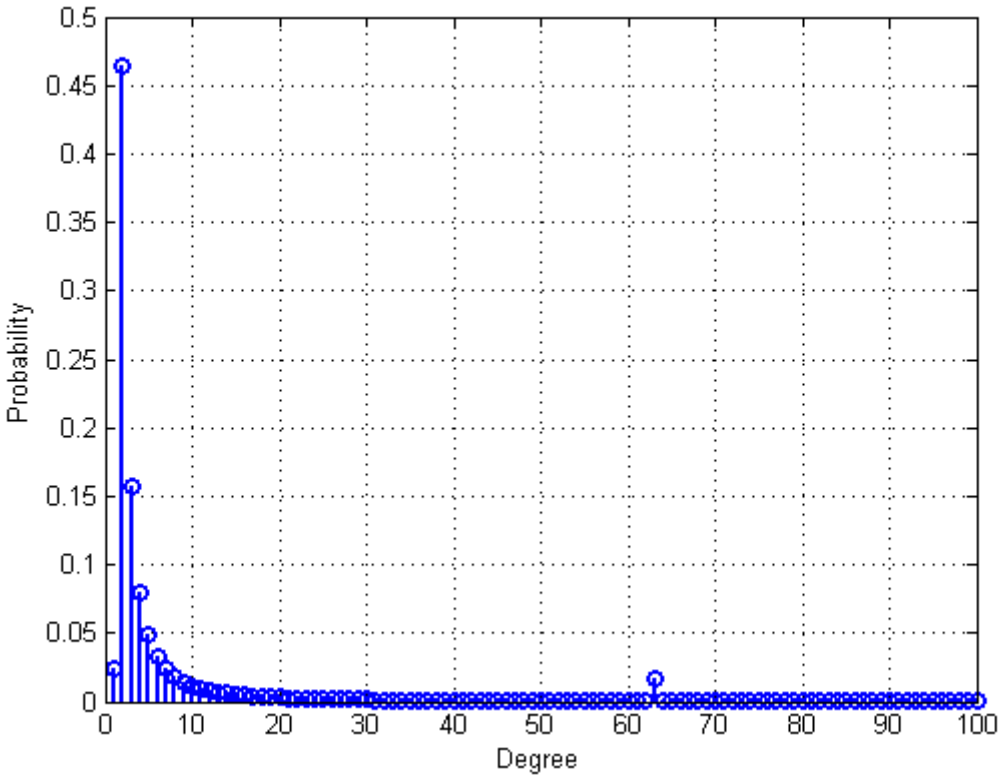

Figure 5.7: Probability mass function of the Robust Soliton distribution for k=100 $\delta = 0.5, c = 0.03$.

threshold P_{suc}; otherwise it is considered as a received packet. If the decoder cannot decode the user message with the available received packet(s), it waits for another received packet to arrive. This process finishes either when the decoder successfully decodes the user data or when the maximum transmission budget is reached.

5.2.2 MATLAB Implementation of the LT Decoding Process

This section explains the LT decoding process used in this project. It is slightly different to the decoding process introduced by Luby [23] which is also described in Section 5.1.4. The decoding process will be introduced by going through a decoding example based on decoding matrix A (shown in Equation 5.1 below). Matrix A shows an optimal case of the LT decoding process – k user packets are decoded with k received packets; there is no overhead. The Roman numbers in Equation 5.1 denote the index of the user packets and Arabic numbers denote the index of the

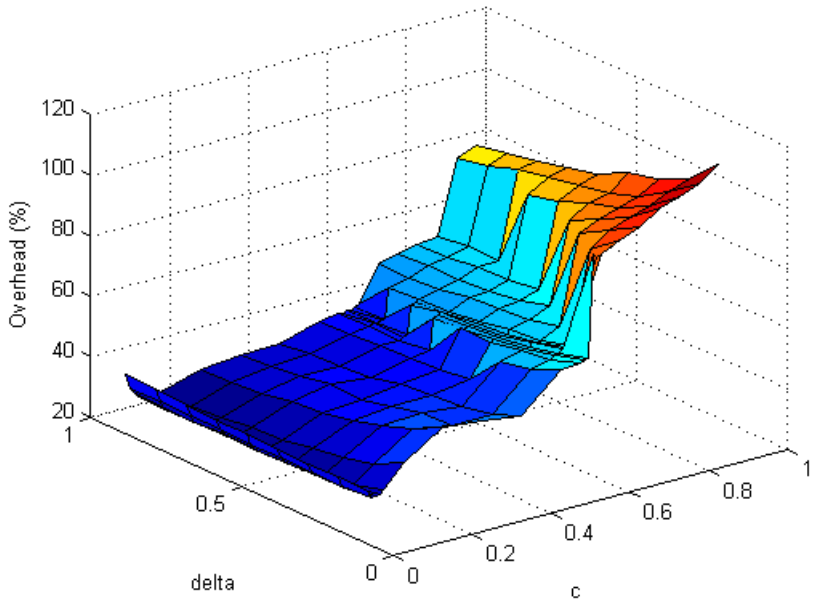

Figure 5.8: The average overhead needed for decoding 100 user packets with different combinations of δ and c.

received packets.

The LT decoding process used in this project contains two stages: starting stage and continuing stage[6]. LT decoding can be started by two types of received packet. The first type of packet is a degree-1 received packet; such a packet indicates straight away the user packet it connects to. The second type of packet is a received packet with a degree of either $w + 1$ or $w - 1$ for which there is a previously received packet with a degree of w such that both packets have w edges mapped to the same user packets. The reception of both packets enables decoding of the user packet that the extra edge links to by XOR'ing the two encoded packets. The name decoding pair is given to such pairs of packets. There are two decoding pairs in matrix A: packets 5 and 8, and packets 2 and 6. The extra user packet V included in packet 5 can be decoded by XOR'ing packets 5 and 8. Using the same principle, user packet VI is decoded by XOR'ing packets 2 and 6. The decoding pairs as well as degree-1 packets can be thought of as the starters of the decoding process.

The next decoding stage is the continuing stage. With the decoded user packets (Packets II and V) from the starting stage, and the edges that link the known user packets and other encoded

[6] This is different to the recover step and process step in Section 5.1.4

packet, more user packets can be discovered. Table 5.1 shows the decoding process of matrix A. It is ordered in the sequence of decoding rather than sequence of packet reception for readers to follow the decoding process easily. The reception sequence is indicated by the first column and the numbers highlighted in blue in the third column indicate the already decoded user packets. This process assumes that all received packets are decoded correctly.

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} I & II & III & IV & V & VI & VII & VIII & IX & X \end{array} \\ \begin{pmatrix} & 1 & & & 1 & & & & & \\ & & & 1 & & & 1 & & & \\ & & & 1 & & 1 & & & & \\ & 1 & 1 & 1 & & 1 & & & 1 & \\ & & 1 & & 1 & & & 1 & 1 & \\ & & & 1 & & 1 & & & & \\ & & 1 & & & & & & & 1 \\ & & 1 & & & & & 1 & 1 & \\ 1 & 1 & & 1 & & 1 & 1 & 1 & 1 & 1 \\ & & & & & & 1 & & 1 & \end{pmatrix} \end{array} \tag{5.1}$$

Table 5.1: An optimal LT encoding case with no overhead, $k = 10$, Robust Soliton distribution $\delta = 0.5, c = 0.03$.

Received Packet No.	Degree	Neighbours	Decoded User Packet	Stage
5	4	3,5,8,9	-	starting
8	3	3,8,9	5	starting
1	2	2,5	2	continuing
2	2	4,7	-	starting
6	3	4,6,7	6	starting
3	2	4,6	4	continuing
2	2	4,7	7	continuing
10	2	7,9	9	continuing
4	5	2,3,4,6,9	3	continuing
5	4	3,5,8,9	8	continuing
7	2	3,10	10	continuing
9	8	1,2,4,6,7,8,9,10	1	continuing

5.2.3 *Validation of the LT code implementation*

The LT decoder built in this project is validated by comparing the overhead distribution for $k = 10,000$ produced by the LT simulation that was built with that from [8] as shown in Figure 5.9. The distribution is found to be similar. The overhead distribution for $k = 10,000$ produced by the LT model built in this project is shown in Figure 5.10. Since there is no statistical information given in [8], no further observations can be made between the two figures. Figure 5.8 shows the overhead for various c and δ also has a similar trend to [18] which is another validation on the implementation.

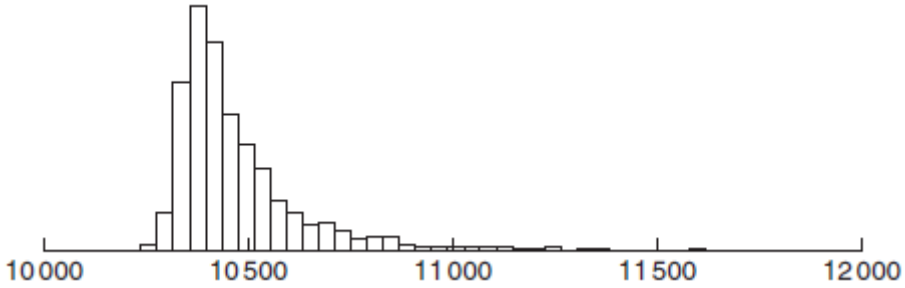

Figure 5.9: Distribution of number of encoded packet needed for decoding 10000 user packets from [8]. δ=0.5 and c=0.03.

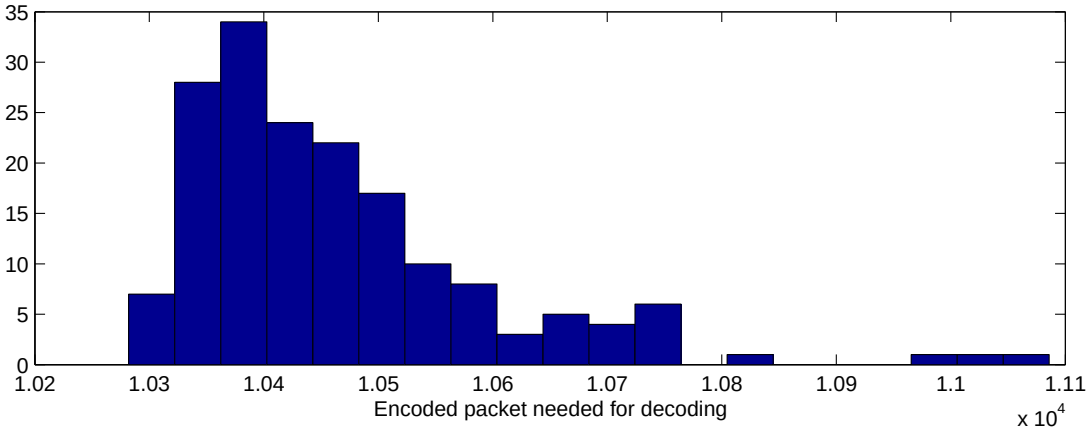

Figure 5.10: Distribution of number of encoded packet needed for decoding 10000 user packets produced by MATLAB simulation built in this project. δ=0.5 and c=0.03.

5.3 Improved LT Codes – LTAM

The asymptotic behaviour in k of LT codes is shown to have an average overhead of less than 5% and it is also reported in [5] as an outstanding advantage of LT codes: the fact that they support independent data reception with very small overhead. However, LT codes was initially designed for data dissemination over the network to solve problems such as mentioned in [34, 39], where the size of data is relatively large – k in the order of 10^4. From the simulations, it was found that for small k values (e.g. $k < 500$), the overhead needed for decoding LT codes is relatively large as shown in Figure 5.11. This high overhead for the small k property of the original LT code is also observed by [4, 22, 53, 44]. Since the user data size in this project has a relatively small k ($k = 86$ as defined in Chapter 2), attention was drawn to the causes of the high overhead. An attempt was made on reducing the amount of overhead needed for decoding in order to better fulfil the aim of the project: to minimise the data reception time by making the data reception process more efficient.

A novel improvement on the LT encoding method has been achieved from this project and the work has been accepted as a conference paper, see [47], in the "2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing". The design details of the improvement are presented in this section.

5.3.1 Causes for High Overhead when k is Small

A large number of instances of LT encoding processes with large overheads was examined and the causes of the large overhead were investigated. It was found that the large overhead is mainly caused by the fact that the user packets are **randomly** selected for each encoded packet. The random mapping nature is one of the key factors in forming the unique feature of the rateless fountain codes: decoding is possible, on average, by receiving **any** $(1 + \epsilon) \cdot k$ encoded packets[7], however it gives no control of the mappings between the user packets and encoded packet, which sometimes gives the following scenarios that cause high overhead for small k:

Scenario 1 One cannot ensure that a particular user packet is included in an encoded packet until the overhead becomes large. In a real simulation example shown in Table 5.2, user packet 2 is not included until the 19^{th} encoded packet is constructed (almost twice the value of k); the overhead reaches as large as 90% for successful decoding due to one missing user packet (user packet 2) in the first 18 received packets.

[7] this enables LT codes to efficiently disseminate data in broadcast transmission.

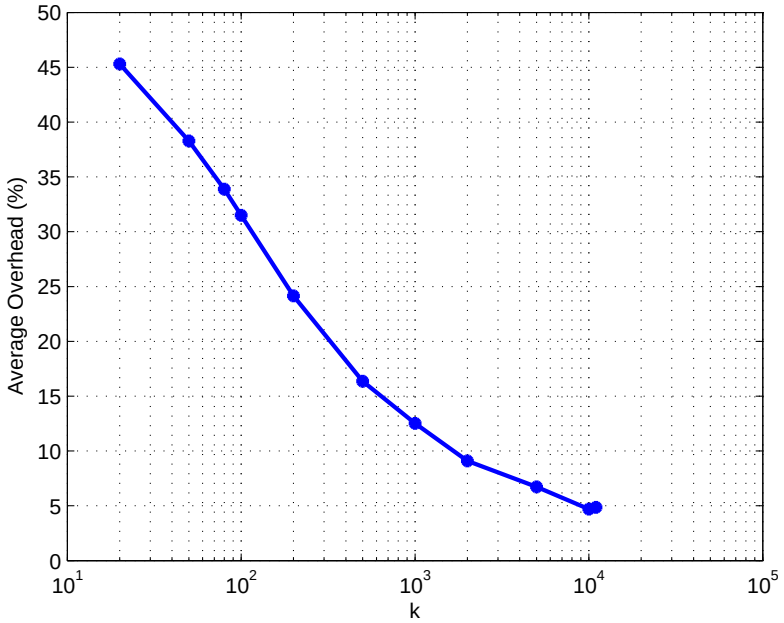

Figure 5.11: Average overhead needed for decoding different length of user data.

Table 5.2 is ordered in sequence of received packet to emphasize how the overhead becomes large during the decoding process. Again, the number highlighted in blue in the third column indicates the already decoded user packets, and the subscripts next to the decoded user packets in the fourth column indicate the received packet the decoding process depends upon to decode the corresponding user packets.

Scenario 2 The edges between the encoded packets and the user packets do not enable the decoding process to complete without a large overhead. An example is shown in Table 5.3. Although all user packets are already included in the first 7 received packets, the mappings between the encoded packets and the user packets do not enable decoding until the 15 encoded packets are received.

In the cases of high overhead, local redundancy also occurs. Local redundant packets[8] are marked with "*" in the fourth column of Tables 5.2 and 5.3. Table 5.2 has nine redundant received

[8] Local redundant packet: a received packet contains user packets that have all been decoded. For example, received packet 12 in Table 5.3 is redundant due to all user packets 1, 2 and 8 are decoded at the reception of encoded packets 3, 8 and 11 respectively. These packets are only locally redundant because different receivers may receive different encoded packets.

Table 5.2: An example of high overhead caused by scenario 1, $k = 10$, Robust Soliton distribution $\delta = 0.5, c = 0.03$. "*" indicates redundant packets.

Received Packet No.	Degree	Linked User Packet	Decoded User Packet
1	2	$3, 10$	-
2	1	6	6
3	2	$8, 10$	-
4	2	$1, 9$	-
5	1	4	4
6	5	1,4,5,7,9	*
7	1	3	$3,10_1,8_3$
8	4	3,7,8,9	*
9	2	6,10	*
10	2	3,9	$9,1_4$
11	2	4,9	*
12	1	8	*
13	2	4,8	*
14	3	3,7,9	7
15	3	1,7,10	*
16	1	9	*
17	5	1,5,6,8,10	5
18	4	5,6,8,10	*
19	2	2,8	2

packets which is 90% redundancy, and Table 5.3 has four redundant packets which is 27% redundancy. While redundancy can be used to combat channel losses (losing redundant packets does not affect the decoding efficiency[9]), it also, in general, delays decoding which increases the data reception time which is unwanted.

This motivates the design of the LTAM method: to decrease the number of received packets for decoding while maximising the LT codes' resilience to various channel losses. The LTAM method is a modification only to the LT encoding process and directly combats the two causes of high overhead for small k mentioned above.

[9] Decoding efficiency – how efficient an LT code can be decoded. It is the number of received packets are needed for successful decoding with respect to k.

Table 5.3: An example of high overhead caused by scenario 2, $k = 10$, Robust Soliton distribution $\delta = 0.5, c = 0.03$.

Received Packet No.	Degree	Linked User Packet	Decoded User Packet
1	3	1,7,8	-
2	2	6,9	-
3	1	1	1
4	2	2,6	-
5	1	10	10
6	1	9	9, 6_2, 2_4
7	7	1,2,3,4,5,6,10	-
8	3	1,2,6	2
9	3	4,7,9	-
10	1	2	*
11	2	4,10	4, 7_9, 8_1
12	3	1,2,8	*
13	3	2,4,10	*
14	3	2,7,9	*
15	3	3,7,8	3, 5_7

5.3.2 Improvement Detail

The two scenarios of large overhead identified above were found to be caused by the random nature of the mappings between the user packets and encoded packets of the original LT codes. The mappings were therefore modified in order to increase the average decoding efficiency. The mappings of certain encoded packets are modified so that they are manually specified in accordance with the mappings of the previously encoded packets. This method requires the encoder to keep the history of encoded packets for the entire period of transmission of a message and it requires memory to record these data, hence the name LTAM is given to this LT encoding method: **LT** codes with **A**dded **M**emory. The LTAM method only modifies the encoding process of the LT codes. The decoding process is exactly the same as that of the original LT codes. The remainder of this section describes the exact operations involved in the LTAM method.

The LTAM method, will be called the LTAM code from now, does not modify the degree distribution. All it modifies are the mappings of the user packets to certain encoded packets. For each new encoded packet, it will be denoted as v, the encoder first draws a degree d from the degree

distribution (i.e. the Robust Soliton distribution). This is independent of all previous packets (this aspect is unchanged). Four different cases are distinguished below. The first three cases deal with the second scenario of high overhead described in Section 5.3.1 – the uncollaborative mappings of the user packets to the encoded packets. The last case deals with the first scenario – the links to one or more user packets are missing.

Case 1 – $d \geq 3$**:** If $d \geq 3$, the encoder makes a random experiment. With probability $1 - p$ the encoder applies the original LT encoding algorithm – picking d randomly chosen user packets and `XOR` them. With probability p the encoder tries to form a decoding pair with a previously encoded packet. This is done by checking the history of the encoded packet to see whether there exists a previously encoded packet of degree $d - 1$. If not, forming a decoding pair fails, and the encoder reverts back to random mapping. If such a packet exists, say packet u, the encoder encodes all $d - 1$ user packets already encoded into packet u, plus one randomly chosen packet in addition for the extra degree into v. The receiver can recover the extra randomly chosen packet if it has received both packet v and u. If there is more than one historic packet of weight $d - 1$, then one is chosen randomly. This case is built to increase the probability of the occurrence of decoding pairs which enable the starting stage of the decoding.

Case 2 – $d = 2$**:** If $d = 2$, then the encoder checks if there are already $k/2$ encoded degree-2 packets constructed. If so, it again makes a random experiment. With probability $1-q$ the mapping is random for packet v. With probability q the encoder searches the history for another packet with degree-2. If it finds none, it again reverts to random mapping. If the encoder finds a previous degree-2 encoded packet u mapped to user data packets x_1 and x_2 say, the encoder selects one of these two randomly (e.g.x_1) for inclusion into packet v, and combines with another randomly chosen packet x_3. With this mapping, by successfully receiving both encoded packets, the recovery of any one of x_1, x_2 or x_3 will trigger the decoding of the other two user packets. The name "continuation pair" is given to such an encoded packet pair. If there are multiple prior existing degree-2 encoded packets, the encoder simply picks a random one to be packet u. The reason that this rule is applied only when there are already $k/2$ encoded degree-2 packets constructed is that this ensures both that there are already sufficient randomly generated degree-2 packets available and the decoding efficiency is not affected too much by the manual mapping[10]. This case increases the probability of the continuation of the decoding process in order to improve the

[10] The effect of manual mapping is explained in Section 5.3.3.

decoding efficiency.

Case 3 – $d = 1$: If $d = 1$, with a probability of q again and when the number of existing degree-2 packets is larger than $k/2$, the encoder checks the history for previous degree-2 packets. If none is found, the encoder chooses a user packet randomly. Otherwise, the encoder selects one of the historic degree-2 packets. The only difference between this case and the previous degree-2 case is, this time if user packets x_1 and x_2 have been encoded into the historic packet u, then the encoder only chooses either x_1 or x_2 for inclusion into the new degree-1 packet v. This can trigger a straight decoding of user packets x_1 and x_2, if both u and v reaches the receiver, which could lead to a new sequence of starting and continuation stages at the receiver. Again, this case increases the probability of continuation of the decoding process.

Case 4 – after $r \cdot k$ encoded packets have been constructed: When the encoder has constructed $r \cdot k$ encoded packets, it checks whether there is any user data packet that has never been encoded before. If so, such a packet is included into the next encoded packet regardless of the degree. This is for solving the problem of large overhead caused by missing user packet(s). One may note that the first three cases are mutually exclusive to one another, but case four is a general strategy that applies to all encoding processes.

Due to the aforementioned four cases, the encoding complexity of the LTAM codes is higher than that of the original LT codes. Because the LTAM encoding process requires a search across all the existing encoded packets for constructing a new encoded packet, the effort is $O(n^2)$ whereas the original LT codes construct each encoded packet independently, the effort is $O(1)$. In the case of unlimited transmission budget, a window can be added to the history look up for the LTAM codes to prevent the encoding process complexity becomes extremely high.

5.3.3 Discussion on Decoding Efficiency

The four cases identified above describe the full LTAM encoding method. One may note that there are various probability checks and constraints for the encoding process to enter the manual mapping process. This is to ensure that the majority of encoded packets contain randomly selected user packets because the channel losses are random; random mapping is the best way to combat the random channel losses. Manual mapping causes the following situation:

1. If the historical encoded packet u that the encoded packet v is based on is lost, packet v automatically loses its meaning. The could potentially decrease the decoding efficiency.

2. Even if both encoded packets in either the decoding pair or continuation pair are received, there maybe a case where the user packet(s) that are guaranteed to be decoded from the pair may already have been decoded; this renders the encoded v a redundant packet. The addition of the probability test also decreases the probability of such situations happening.

The constraints to enter the manual mapping mode are to ensure that the advantage of manually specifying the encoded packets is utilised while the disadvantage of it is minimised and hence the decoding efficiency of the LTAM method is maximised.

5.3.4 The Best Parameters for LTAM

There are various parameters for the LTAM method: parameters inherited from the original LT codes such as δ and c used in the Robust Soliton distribution, and LTAM's own parameters such as p, q and r. Different combinations of parameters of LTAM codes were examined in terms of the overhead produced in order to find the configurations that give the best overall decoding efficiency. They were tested under different channel conditions as well as with different message lengths for completeness.

In order to make this task feasible within the time limit of this project, only experiments that were considered important and relevant were conducted. The total number of experiments grows as the number of parameters increases and can be formulated as $\prod_{i=1}^{n} x_i$ where x_i are the resolutions of the i^{th} parameter (number of values tested for the corresponding parameter), and n is the total number of parameters.

After a number of experiments, the suitable values for the parameters of the LTAM code were determined and the chosen parameter values for LTAM are shown in Table 5.4. The experimental results are presented in the first three sections of Appendix B along with the explanations for these selections. The tabulated parameter values are used throughout this thesis unless explicitly specified otherwise.

5.3.5 A Failed Attempt on Improving LT Codes

During the development of the LTAM method, another method was proposed and was found to have performance that is worse than the original LT codes. Since the method seems to be reasonable, the author decided to document it in this thesis for future reference.

In this method, the degree distribution was also not modified. The number of inclusion of a user packet to an encoded packet is decided to be kept uniform across all k user packets. The

Table 5.4: Values of the parameters that are decided to be used for the LTAM codes.

Parameter Name	Chosen value
Degree Distribution	Robust Soliton
δ	0.9
c	0.01
p	0.6 for $k = 20$ 0.3 for $k = 86$ 0.2 for $k = 200$ 0.1 for $k = 500$ 0.05 for $k > 500$
q	0.5
r	0.97

user packets that have smaller number of inclusions than others will be selected as one of the neighbour(s) for the next encoded packet. However, this method was found to be not working as well as the original LT codes. The main reasons are:

- The decoding of a user packet does not depend on the number of times the user packet is included into the encoded packets, it depends on the relationship between the user packet and other user packets in the encoded packets.

- The channel loss pattern is random and unknown. Comparing with manually controlling the number of inclusions of user packets, random mapping has a better chance of getting the message through earlier.

5.4 Performance Comparison between Different LT Codes

In this section, the overhead comparison among the original LT code, the LTAM code and a "sub-optimal" method[11,12] is presented. The comparisons for the three schemes are done under no loss

[11] This is a previously published improvement of LT codes presented in [53]. A "sub-optimal" degree distribution was designed and it was shown to require less overhead for decoding in [53]. The results for this method are obtained from the simulation model built according to method's description presented in their paper.

[12] The overhead performance shown in [53] is similar to the sub-optimal simulation built according to the paper for $k < 1000$, however [53] shows a better performance for $k > 1000$. It may be caused by different values used for several unclear described parameters that are presented in the paper. Nevertheless, the LTAM code outperforms the better sub-optimal method shown in the paper for $k > 1000$.

channels, BEC channels and GE channels for k up to 10,000. Since the encoded packets produced by both the original LT code and the sub-optimal method are independent from one another, different channel loss rates do not affect their overhead performance. However, the overhead performance of the LTAM code does depend on channel loss patterns due to the dependencies among certain LTAM encoded packets. Figure 5.12 shows the overhead comparison under no loss channels and Figure 5.13 shows the overhead comparison under BEC and GE channels.

Figure 5.12 shows that under no loss channels, the LTAM code performs better than both the original LT code and the sub-optimal method for $k < 10000$. The LTAM code has as much as about 10% lower overhead than the original LT code for $k \leq 100$, and an average of 3% lower overhead than the sub-optimal method for $k \leq 200$. Figure 5.13 shows the comparison of the original LT code, the sub-optimal method with the performance of the LTAM code under BEC with $PER = 0.5$ and GE channel conditions with $p_{b,g} = p_{g,b} = 0.1$, $PER_b = 0.5$ and $PER_g = 0$. From Figure 5.13 it can be seen that the LTAM code has a certain degradation under lossy channels; the most obvious point is the LTAM performance under GE channel with $PER_b = 0.9$ for $k = 20$. The difference between the LTAM code and the sub-optimal method is very small. However, the LTAM code still performs better than both the original LT code and the sub-optimal method for all k tested under lossy channels.

5.5 Conclusion

This chapter introduced the working principle of LT codes, identified their problem of high overhead for short length message transmissions, and described an improvement to the high overhead of LT codes, namely the LTAM codes. The LTAM codes neither modify the degree distribution of the encoding process nor the decoding process. It only changes the way user packets are selected for an encoded packet in order to enable faster decoding at the receiving ends. The LTAM codes proposed are shown to have fulfilled the aim that it is designed for from simulations – a smaller overhead is achieved for $k < 10000$ than the original LT codes. It is also shown to have outperformed another improvement on the LT codes for short message lengths.

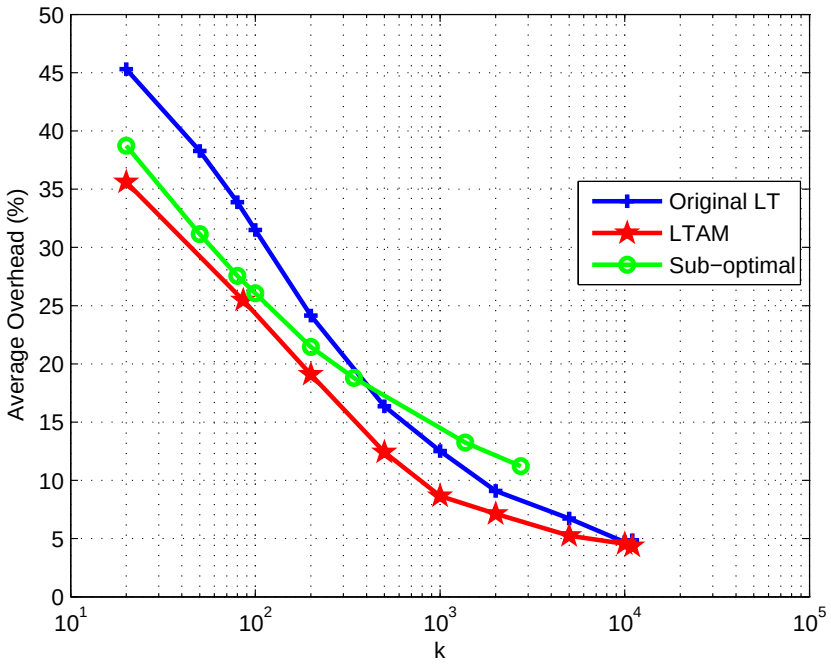

Figure 5.12: Comparison between three LT schemes under no-loss channel.

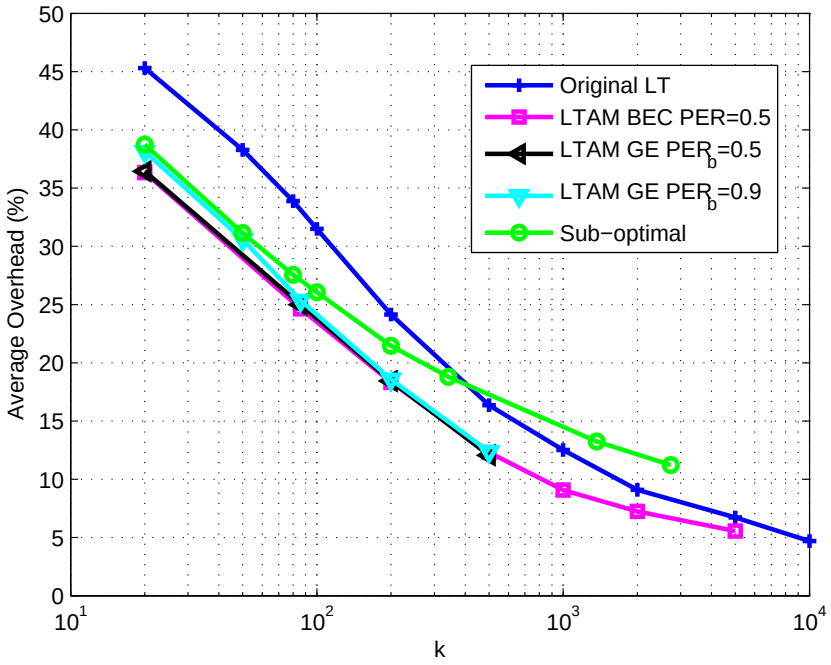

Figure 5.13: Comparison among three LT schemes under the two channel models.

Chapter VI

Performance Comparison

This chapter compares the performance of the four coding schemes (the repetition codes, the RS codes, the original LT codes, and the LTAM codes) according to the two performance measures defined in Chapter 2: the success rate and the transmission cost. This is done for the two channel models – memoryless channels (BSC) and burst error channels (GE), and two code rates: 1/2 and 1/3. Only one user data size and packet size were used and they were specified in Tables 2.1 and 2.2 in Chapter 2. This chapter contains five main sections: Sections 6.1 and 6.2 describe and explain the performance comparisons for the success rate, and the transmission cost respectively. Section 6.3 describes the scheme that combines RS codes with LT codes and shows the performance of this new scheme. This scheme is proposed in response to the significant performance difference (as shown and explained in Section 6.1) between the LT codes and the RS codes due to the packet erasure property[1] of the LT codes. Section 6.4 presents a top level comparison of the decoding complexity of the candidate schemes. Finally Section 6.5 concludes the chapter.

The performance comparisons are presented as 2D graphs. For memoryless channels, the abscissa shows the channel error rate. For burst error channels, the abscissa shows the channel error rate of the "bad" state of the GE model. The ordinate shows either the success rate or transmission cost. In each performance comparison graph, there are either four or three curves that show the performance of the candidate coding schemes. The graphs that have all four curves are comparisons of the four schemes for a range of BERs and the graphs that have three curves are comparisons of the three packet erasure schemes (the repetition codes and the two LT codes) for a range of PERs. It was more appropriate to present the performance graphs for the three packet level erasure coding schemes at various PERs rather than BERs. However it seemed inappropriate to show the RS codes at different PERs since the error correction performance of the RS codes is irrelevant to the PER; it was used as an intra-packet coding scheme. Therefore the following arrangement

[1] Packet erasure property: only packets received correctly are going through the decoding process. There is no mechanism of fixing a corrupted packet. The probability of correctly receiving packets is limited by the BER and the packet length.

was made: the performance of all four coding schemes is compared first at various BERs just to show the performance difference between RS codes and the other three schemes. The comparison among the three packet level erasure coding schemes is then presented at different PER.

There are 16 graphs in total to show the performance comparison among the four coding schemes under the two performance measures, the two channel models, the two code rates and the two abscissa types. Table 6.1 shows a summary of the figure numbers and page numbers of the figures grouped into their corresponding channels, code rates and abscissa types (either BER or PER) for quick reference.

Table 6.1: A summary of figure numbers that show the performance comparison for the proposed coding schemes.

Channel	Code Rate	Measure	BER	Page No.	PER	Page No.
BEC/BSC	$1/2$	Success Rate	6.1	53	6.3	54
		Transmission Cost	6.2	53	6.4	54
	$1/3$	Success Rate	6.5	55	6.7	56
		Transmission Cost	6.6	55	6.8	56
GE	$1/2$	Success Rate	6.13	63	6.15	64
		Transmission Cost	6.14	63	6.16	64
	$1/3$	Success Rate	6.17	65	6.19	66
		Transmission Cost	6.18	65	6.20	66

6.1 Success Rate

This section describes the success rate performance of the coding schemes for both memoryless channels and burst error channels. Subsections 6.1.1 and 6.1.2 describe the performance under the memoryless and burst error channels respectively. For each channel model, the performance for code rates 1/2 and 1/3 is presented. Subsection 6.1.3 discusses the results.

6.1.1 Results: Memoryless Channels

As shown in Figures 6.1 (pg.53) and 6.5 (pg.55), the success rate curves of the four coding schemes have the same relative order under memoryless error channels for both code rates: the RS code

has the best performance, the LT codes[2] have the second best performance (with the LTAM code performing slightly better than the original LT code), and the repetition code has the worst performance.

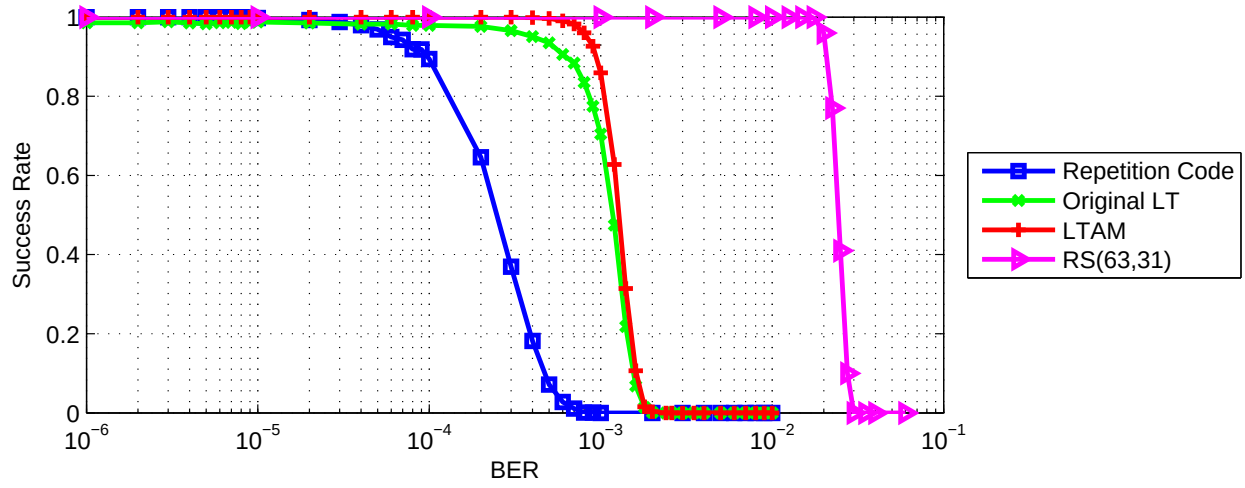

Figure 6.1: Success rate, memoryless channel, $k = 86, j = 378, R_c = 1/2$. All graphs in this chapter are default to $k = 86, j = 378$.

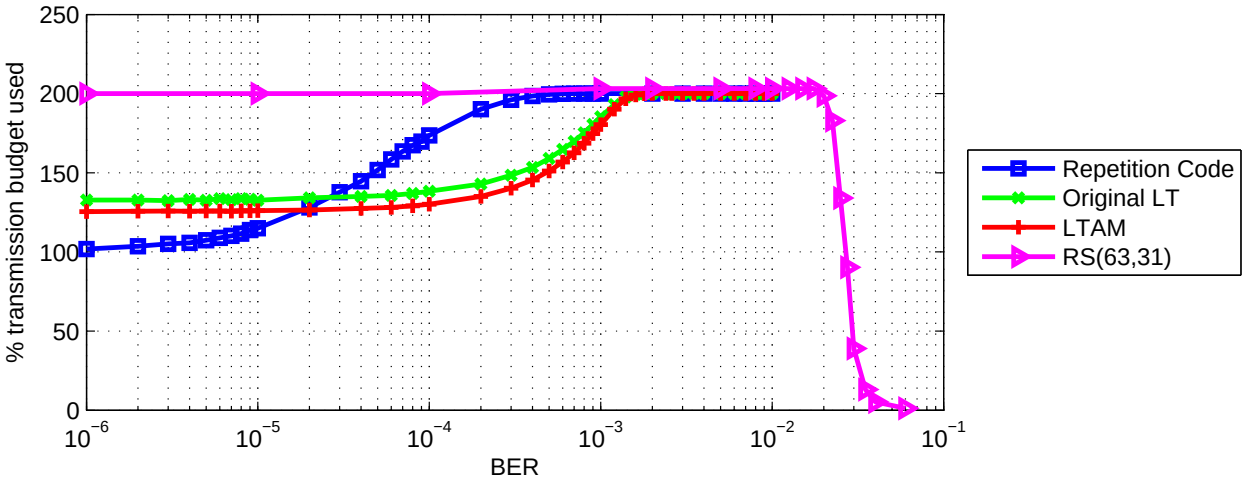

Figure 6.2: Transmission Cost, memoryless channel. $R_c = 1/2$.

[2] The term LT codes will be used as a collective phrase to indicate the LTAM code and the original LT code.

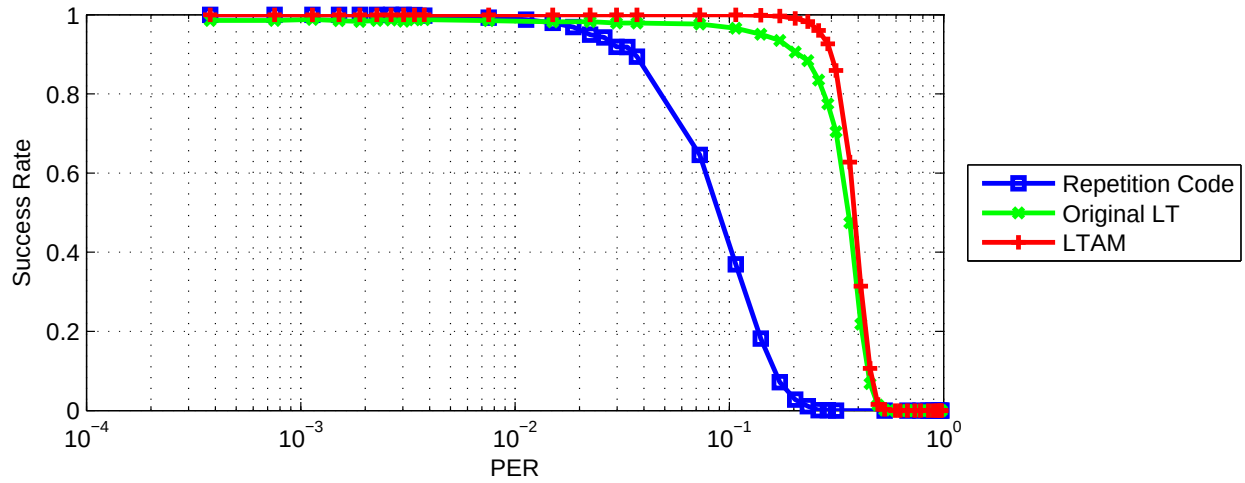

Figure 6.3: Success rate at different PER, memoryless channel, $R_c = 1/2$.

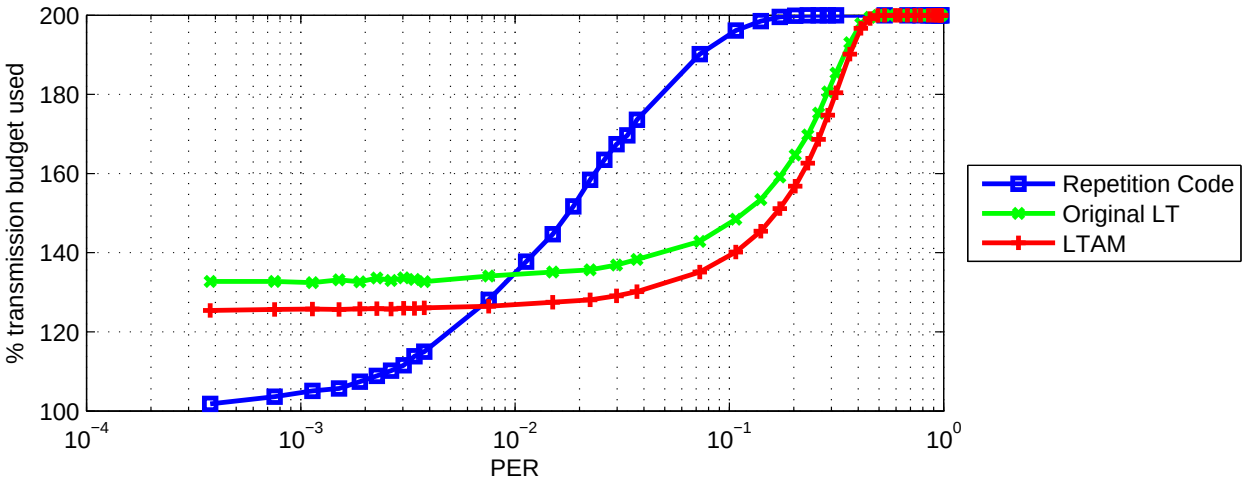

Figure 6.4: Transmission cost at different PER, memoryless channel, $R_c = 1/2$.

Table 6.2 summarises the channel BER and PER values at which each coding scheme starts exhibiting significant errors. The relevant figures are 6.1 (pg.53) and 6.3 (pg.54) for $R_c = 1/2$ and 6.5 (pg.55) and 6.7 (pg.56) for $R_c = 1/3$. By decreasing the code rate from 1/2 to 1/3, every coding scheme shows a certain improvement in the abscissa value of the knee[3] which indicates the highest channel error rate that the coding scheme can tolerate or keep almost 100% success

[3] the abscissa point where the success rate curve starts to drop significantly

rate. The LTAM code has better performance than the original LT code for both code rates as shown in Figures 6.3 (pg.54) and 6.7 (pg.56). For $R_c = 1/2$, the LTAM code has at least a 10% higher success rate than the original LT code during the transition interval[4] and the difference reaches a maximum of 15% between PER=0.3 and 0.36. For $R_c = 1/3$, the performance of LTAM outperforms the original LT codes by as much as 20% during the transition interval.

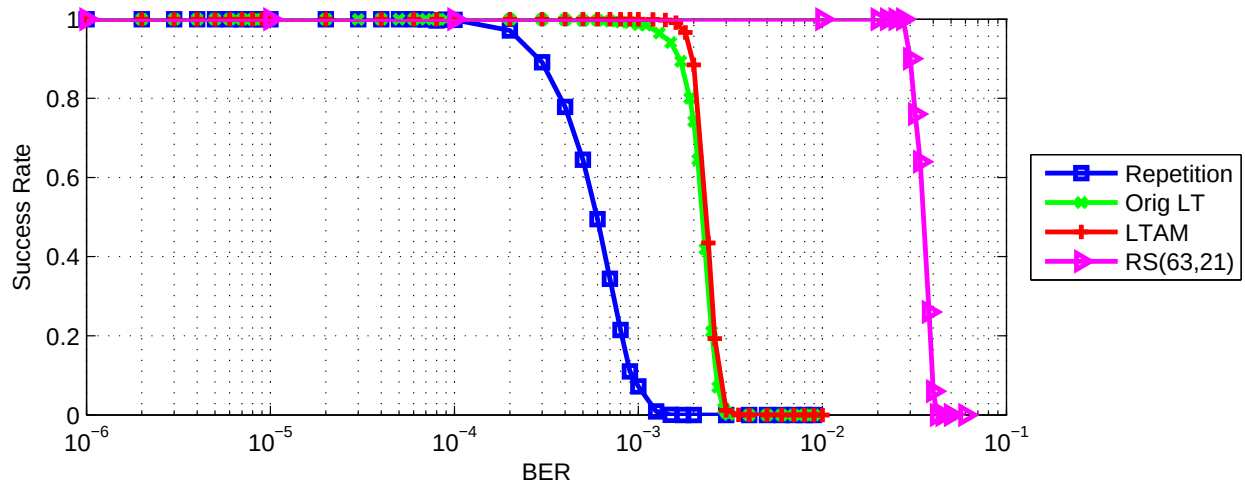

Figure 6.5: Success Probability, memoryless channel, $R_c = 1/3$.

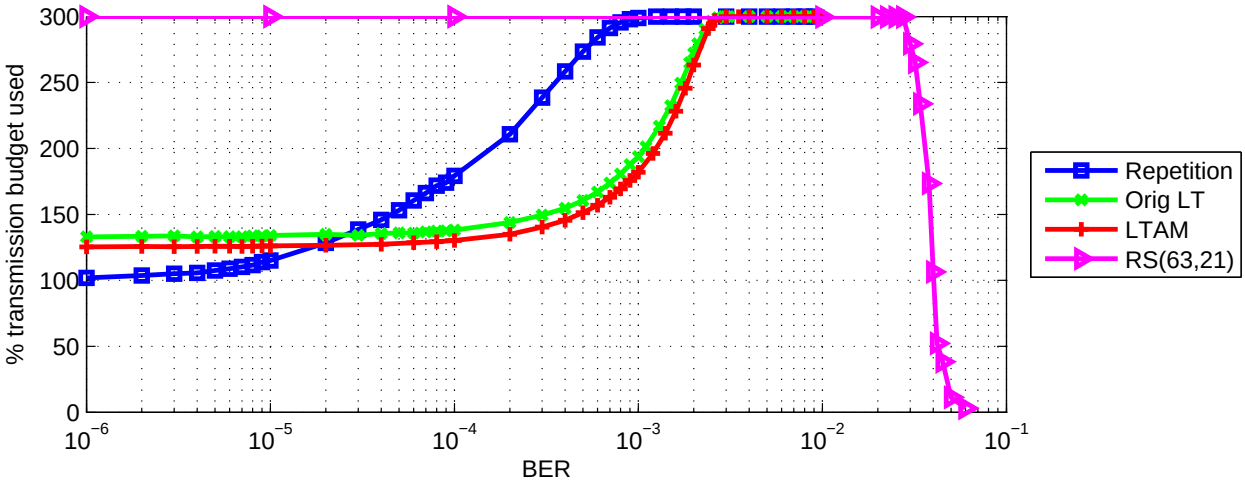

Figure 6.6: Transmission cost, memoryless channel, $R_c = 1/3$.

[4] The range of channel error rate that the success rate transits from a stable high value to a low value.

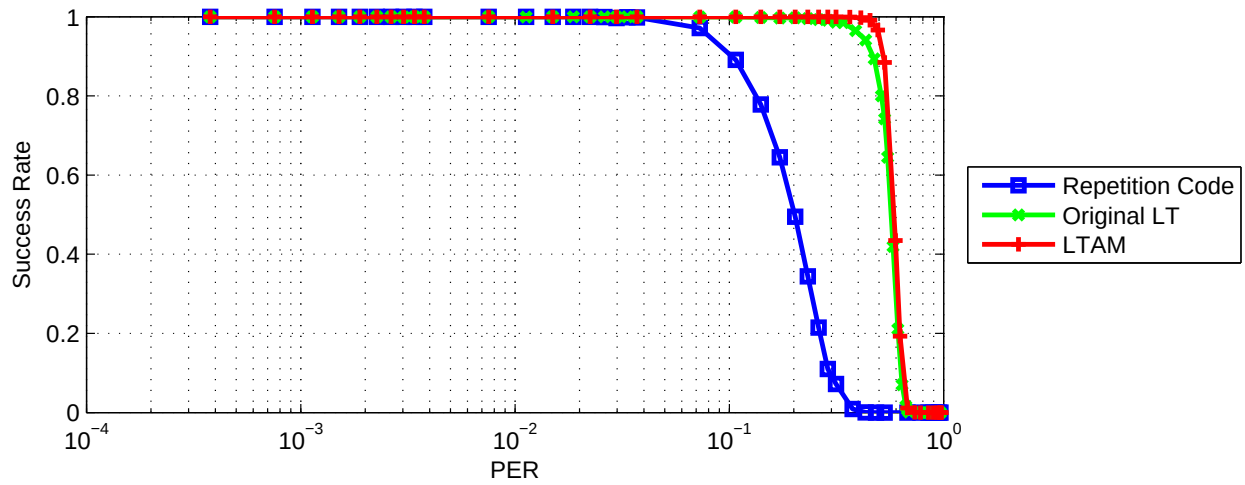

Figure 6.7: Success rate at different PER, memoryless channel, $R_c = 1/3$.

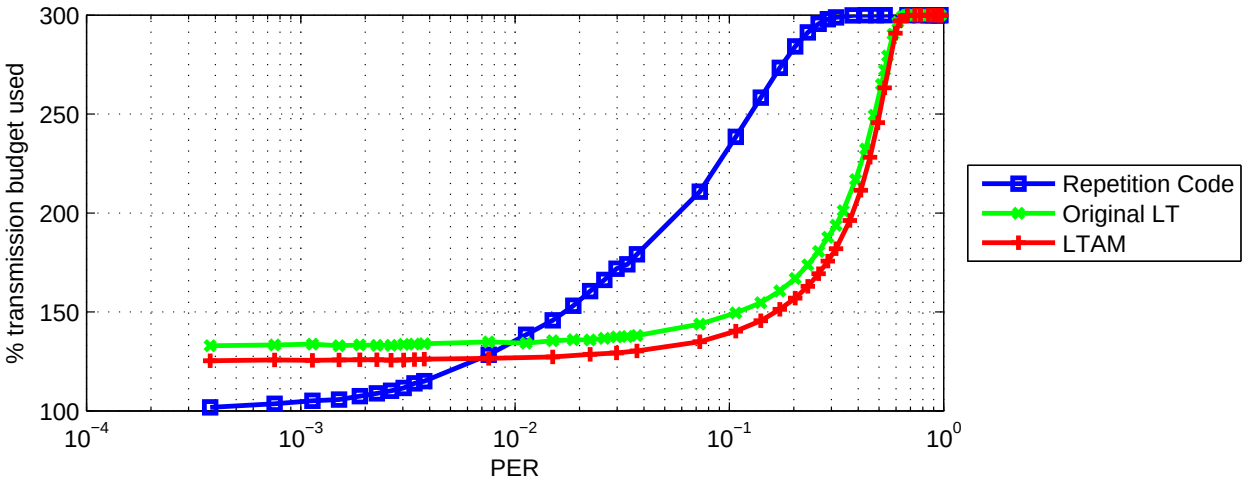

Figure 6.8: Transmission cost at different PER, memoryless channel, $R_c = 1/3$.

6.1.2 Results: Burst Error Channels

The success rate performance of the four coding schemes over the GE channel follows the same general order as in the case of memoryless error channel except: as BER_b increases, both types of LT codes eventually outperform the RS code for both $R_c = 1/2$ and $1/3$. Nevertheless, the knee of the RS code is still about one order of magnitude more than that of the LT codes for both code rates.

For $R_c = 1/2$, the success rate of the RS code starts to drop from BER_b=0.02 and reaches

Table 6.2: Channel error rates at which success rates fall significantly below 100%.

Code Rate	Scheme	Channel BER	Channel PER
$1/2$	Repetition	–	0.01
	Orig LT	–	0.1
	LTAM	–	0.2
	RS	0.02	–
$1/3$	Repetition	–	0.07
	Orig LT	–	0.4
	LTAM	–	0.5
	RS	0.03	–

zero success rate at BER_b=0.04. The LT codes cross the RS code at BER_b=0.03 after which the LTAM code plateaus at success rate=0.17 and the original LT code plateaus at success rate =0.13. For $R_c = 1/3$, the RS code starts to drop at BER_b=0.03 and reaches zero at BER_b=0.04. Both LT codes this time outperform the RS code from BER_b=0.035 and retain success rates as high as 0.8 for the LTAM code and 0.7 for the original LT code up to the highest BER_b value (0.5).

The transition interval of the repetition code is between PER_b=0.02 and 0.9 for $R_c = 1/2$ and between PER_b=0.07 and 1 for $R_c = 1/3$. The repetition code also never decreases to zero with high BER_b when $R_c = 1/3$ as shown in Figure 6.17 (pg.65). This suggests there is a certain successful decoding probability (4.9% as shown in the graph) from receiving packets during the time that the channel is in the good state.

6.1.3 Discussion

A Large Performance Difference between the RS Codes and the LT Codes

The channel error rates of the knees between the LT codes and the RS codes differs as large as about one order of magnitude. This large difference is mainly caused by the packet erasure nature of the LT codes – unless every bit in a packet is successfully received, the packet is erased. There is no mechanism to fix an erroneous packet in the LT code implementation. The performance of the LT code is limited by the probability of successful reception of a packet, which is severely limited by the BER of the channel, as shown in the relationship graph in Figure 6.9 (pg.58). A scheme that combines an RS code with the LT code is proposed in order to increase the probability of successful reception of a packet, hence the overall success probability of the LT codes. This scheme is presented in Section 6.3.

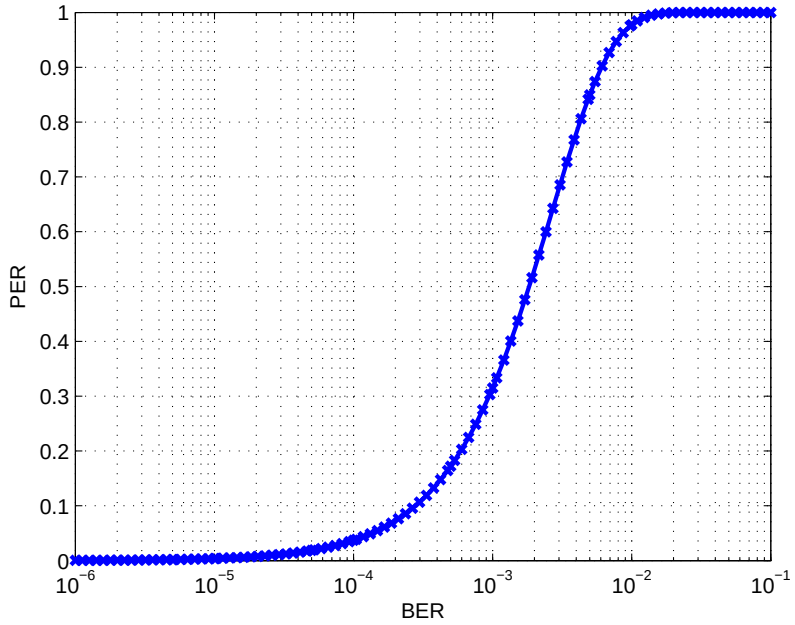

Figure 6.9: Relationship between Bit Error Rate (BER) and Packet Erasure Rate (PER) packet size, $j = 378$.

The Original LT Code Never Reaches 100% Success Rate for R_c=1/2

The original LT code never reaches a 100% success rate no matter how small the channel error rate is for both channel models. This is most obvious in the figures that show the success rate performance for code rate 1/2 (e.g. Figure 6.3 (pg.54)) and less obvious for that of code rate 1/3 since the original LT code success rate is much closer to 1 at small BER values for $R_c = 1/3$. The success rate of the original LT code is bounded away from one is due to the fact that the distribution of the overhead of the original LT code spreads beyond twice the value of k even over loss-free channels as shown in Figure 6.10 (pg.59). This causes an average success rate less than 100% for $2 \cdot k$ budget even for channels with very small loss rates.

Figure 6.10 shows the distribution of the number of received packets needed for decoding 86 user packets under perfect channel conditions (no packet loss) for the original LT encoding method with 10,000 experiments. For $R_c = 1/2$, the available budget is 172 encoded packets for $k = 86$. This is indicated as a red bar with a square tip in both Figures 6.10 and 6.11 (pg.60) (where Figure 6.11 shows the same distribution for the LTAM code). Let n denote the number of received packets needed to decode the user data for an experiment. From Figure 6.10, it can be seen that n can be

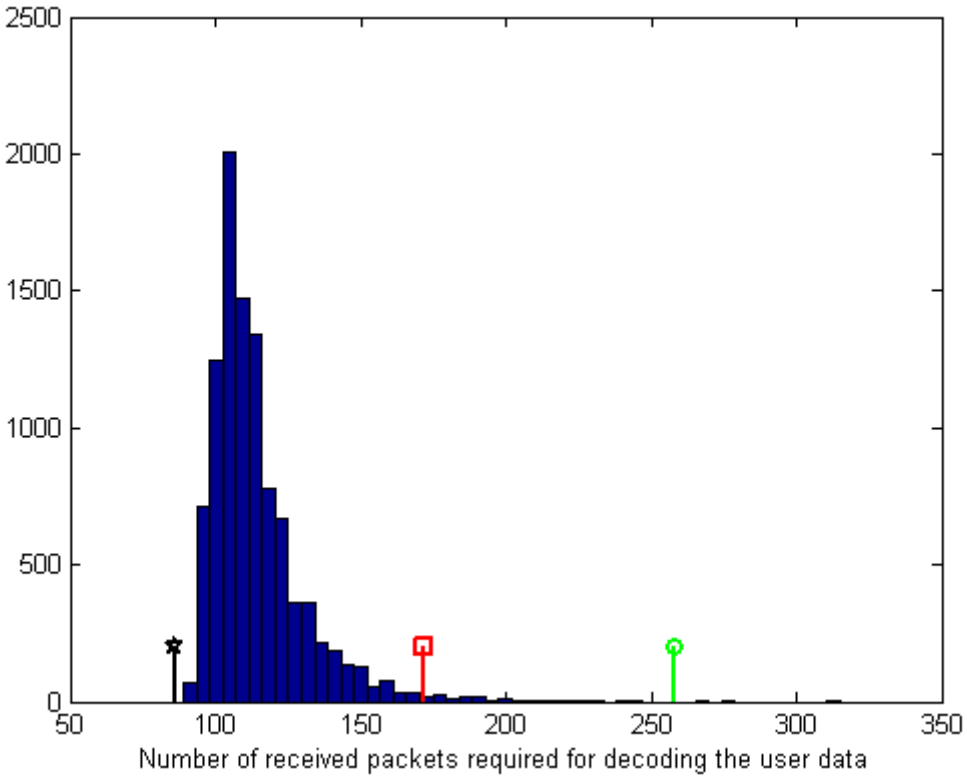

Figure 6.10: Distribution of the numbers of encoded packets to decode the original LT code under loss-free channels with 10,000 experiments, $k = 86$. The black bar with a star tip indicates k packets, the red bar with a square tip indicates $2 \cdot k$ packets and the green bar with a circle tip indicates $3 \cdot k$ packets.

as high as 315; it is about 3.7 times more than k. Although it only occurred once in 10,000 experiments, 131 experiments among the 10,000 experiments where n was exceeded a budget of $2 \cdot k$. This corresponds to 1.3% of the total number of experiments. This explains the fact that the original LT code only has a maximum success rate of about 98.7% for $R_c = 1/2$.

Figure 6.11 shows the distribution of n for $k = 86$ under loss-free channel for the LTAM code. Again 10,000 experiments were conducted. The LTAM code had much better statistics than the original LT code. Among 10,000 experiments, only 4 cases exceeded the total budget of $2 \cdot k$ (these only contributed to 0.04% of the total experiments conducted) and the maximum n occurred is only 184 rather than 315 in the case of the original LT code. From these graphs (Figure 6.10 and Figure 6.11), the improvement made by the LTAM code to the original LT code can be seen.

For $R_c = 1/3$, the total budget is shown as a green bar with a circle tip in Figures 6.10 and

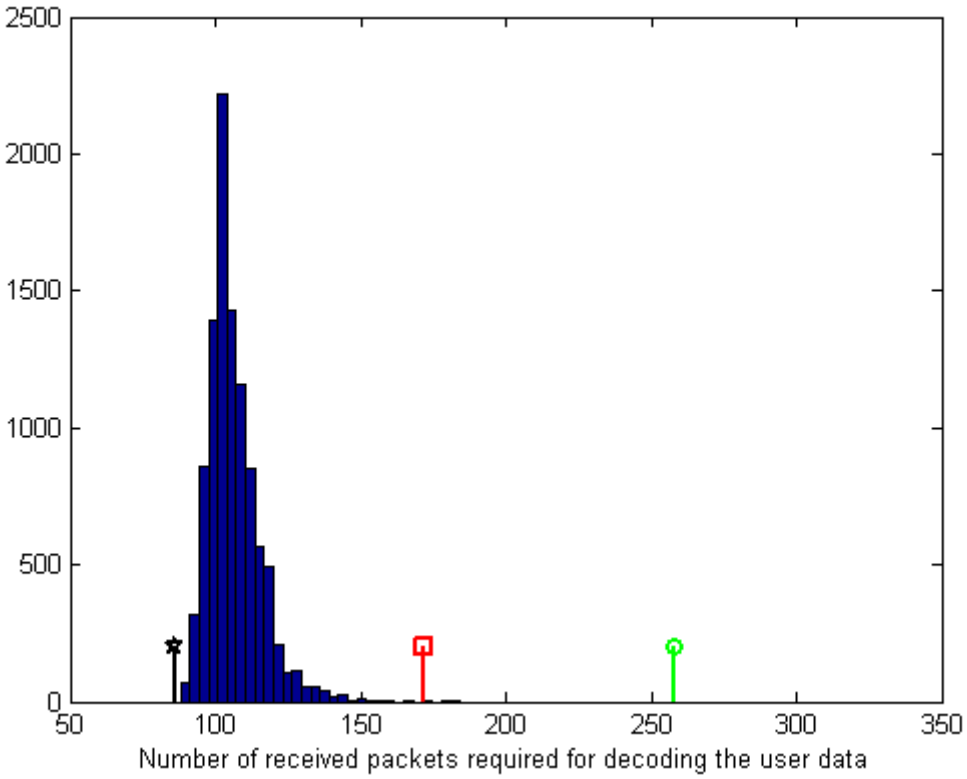

Figure 6.11: Distribution of the numbers of encoded packets to decode the LTAM code under loss-free channels with 10,000 experiments, $k = 86$. The black bar with a star tip indicates k packets, the red bar with a square tip indicates $2 \cdot k$ packets and the green bar with a circle tip indicates $3 \cdot k$ packets.

6.11, there were three cases that the original LT code exceeded the total budget and there was no case that the LTAM code exceeded the $3 \cdot k$ budget.

Figure 6.12 shows box plots for the distributions shown in Figures 6.10 and 6.11. The lower and upper horizontal edges of the box show the 25% and 75% quantile respectively, the red horizontal bar in the box is the median and the black horizontal bars below and above the box are the lower and upper adjacent respectively. The red crosses above the upper adjacent are the outliers. From this figure, the improvement made by the LTAM code can be seen. Table 6.3 tabulates the important statistical information for Figure 6.12.

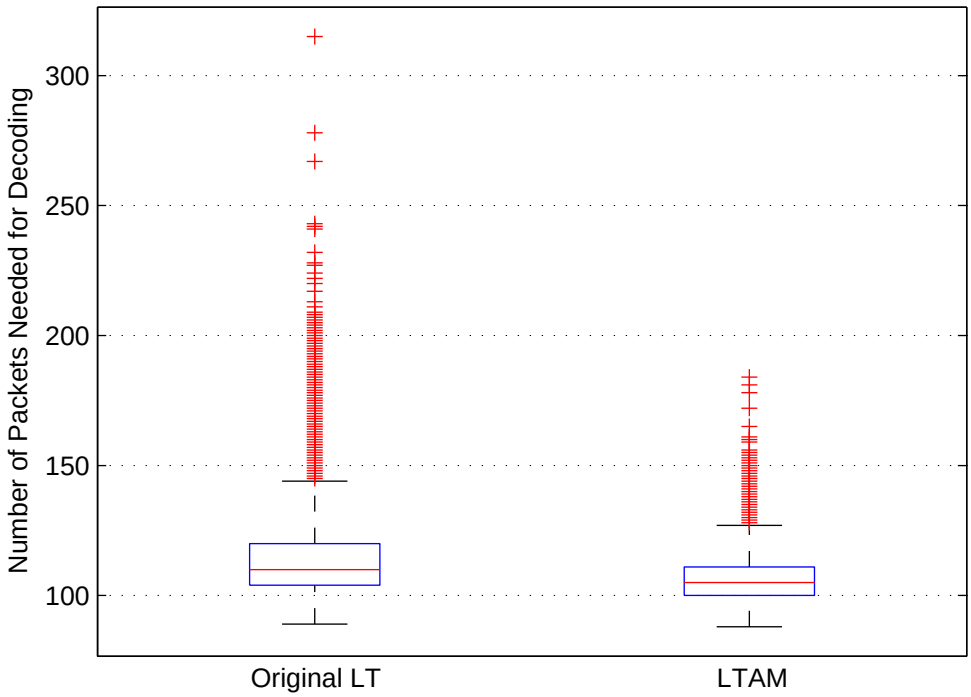

Figure 6.12: Statistical comparison between the original LT code and the LTAM.

Table 6.3: Important statistical information for Figure 6.12.

	Number of received packets to decode		
	Original LT	LTAM	Difference
Maximum	315	184	131
Upper Adjacent	144	127	17
75% Quantile	120	111	9
Median	110	105	5
25% Quantile	104	100	4
Lower Adjacent	89	88	1
Minimum	89	88	1

The Success Rate of the LT Codes are Bounded Away from Zero over the GE Channels for all BER_b Tested

The performance of all coding schemes over the GE channel is only tested for a range of BER_b values; both self transition probability $p_{g,g}$ and $p_{b,b}$ were fixed to 0.9. The value of BER_g was

held to zero. Because of this, for all GE channels tested, the mean PERs do not reach zero and neither do the success rates of the LT codes as shown in Figures 6.13 (pg.63) and 6.17 (pg.65). The worst BER_b tested is 0.5 (PER_b=1) which gives a worst mean PER of 0.5 for a symmetrical GE channel. Unlike the RS codes, where the success rate is dependent on the length of the burst errors, the success rate of the LT codes is only affected by the average error rate and not by the pattern of the errors.

This can be seen from the two figures 6.13 and 6.17: once $BER_b = 0.02$ or $PER_b = 1$ which indicates the mean PER of the GE channel is 0.5, the performance of the LT codes plateaus to stable success rates. The original LT code and the LTAM code levels off at success rates of 12% and 17% for $R_c = 1/2$ and exceptionally high success rates of 72% and 81% respectively for $R_c = 1/3$.

RS Codes have a Sharp Drop-off

RS codes have an error correction capability of t symbol errors per packet. Once there are more than t errors per encoded packet, the success probability for decoding RS(63,21) and RS(63,31) codes are zero[5]. Hence, once the error rate exceeds the threshold of t symbols per packet, it is expected that the success rate would drop to zero quickly.

[5] This is specifically mentioned for RS(63,21) and RS(63,31) because for certain RS codes there is a relatively low probability that the message can be decoded with more than t symbol errors. For example RS(15,7), from experiments, there is 1.4% probability that the decoding is successful with $t + 1$ symbol errors. For RS(15,9), there is 1% success decoding probability with $t + 1$ errors.

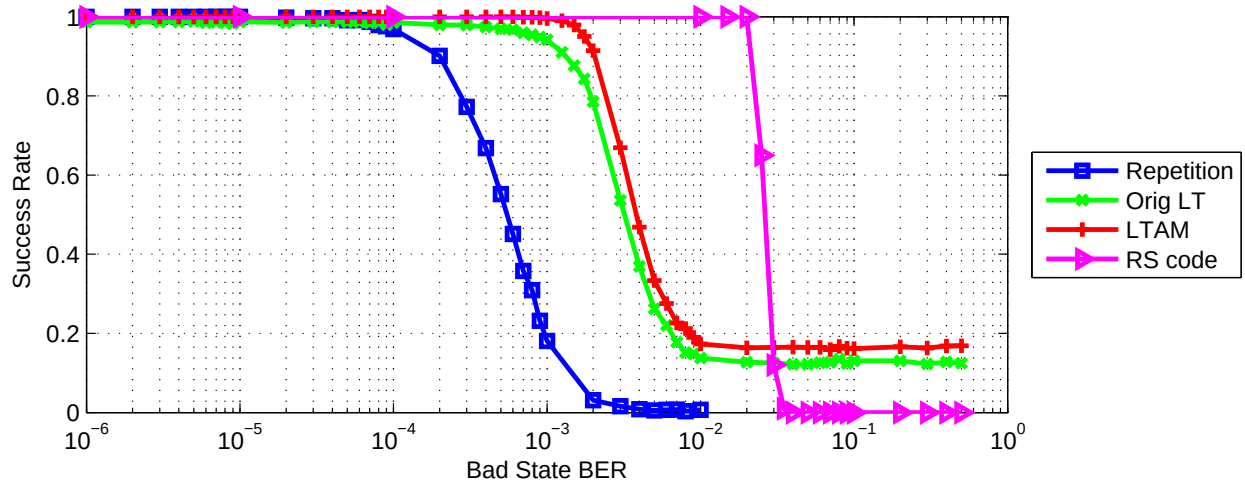

Figure 6.13: Success rate, Gilbert-Elliott (GE) channel, $R_c = 1/2$.

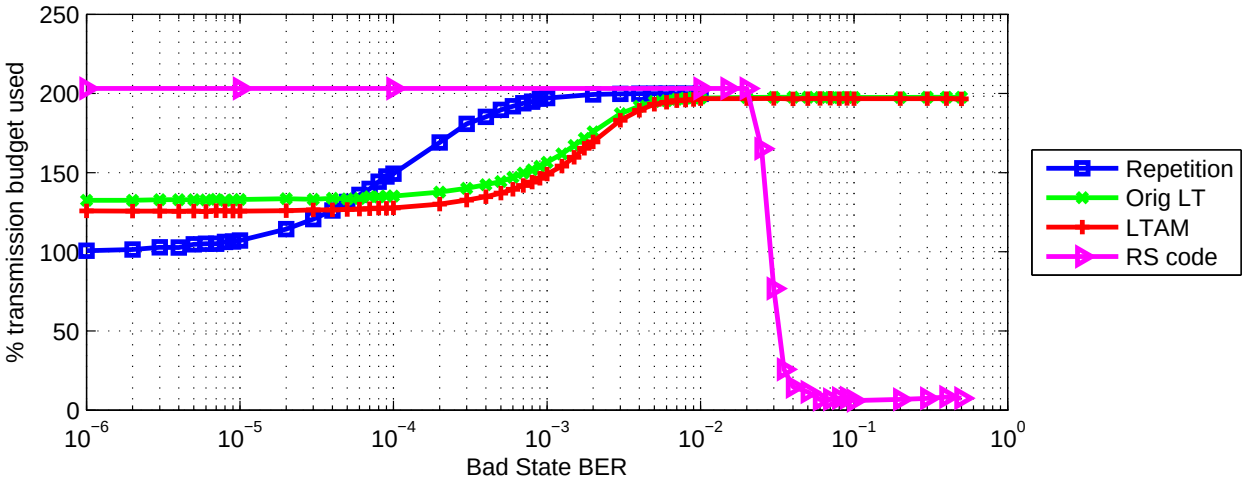

Figure 6.14: Transmission cost, GE channel. $R_c = 1/2$.

6.2 Transmission Cost

In the previous section, the success rate performance of all candidate coding schemes under various channels and code rates is described and discussed. This section focuses on presenting the transmission cost for the coding schemes. The figures showing the performance comparisons in transmission cost are Figures 6.2, 6.4, 6.6, and 6.8 (pg.53, 54, 55, and 56)for memoryless error

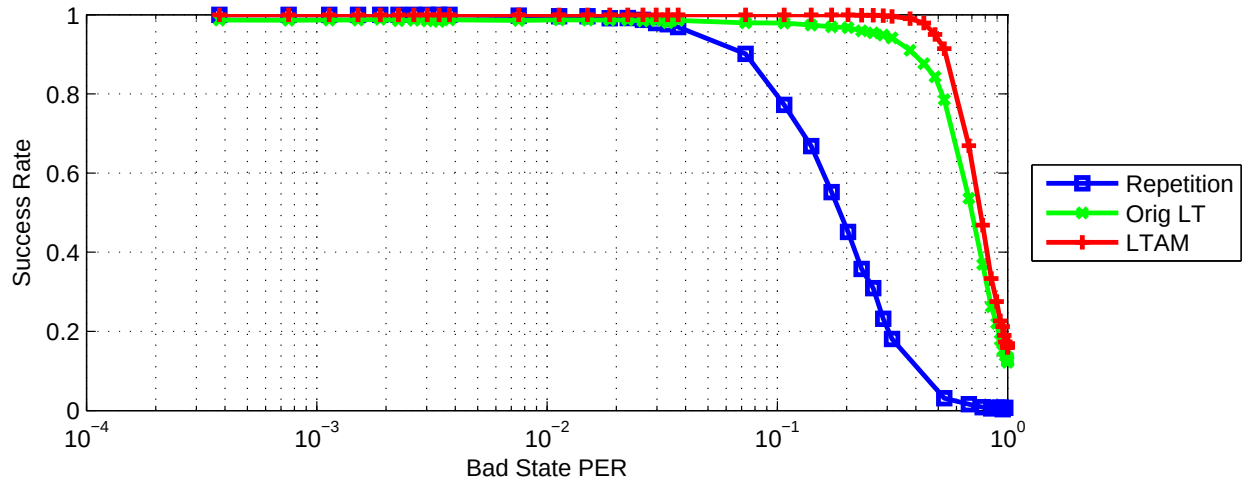

Figure 6.15: Success rate at different PER, GE channel, $R_c = 1/2$.

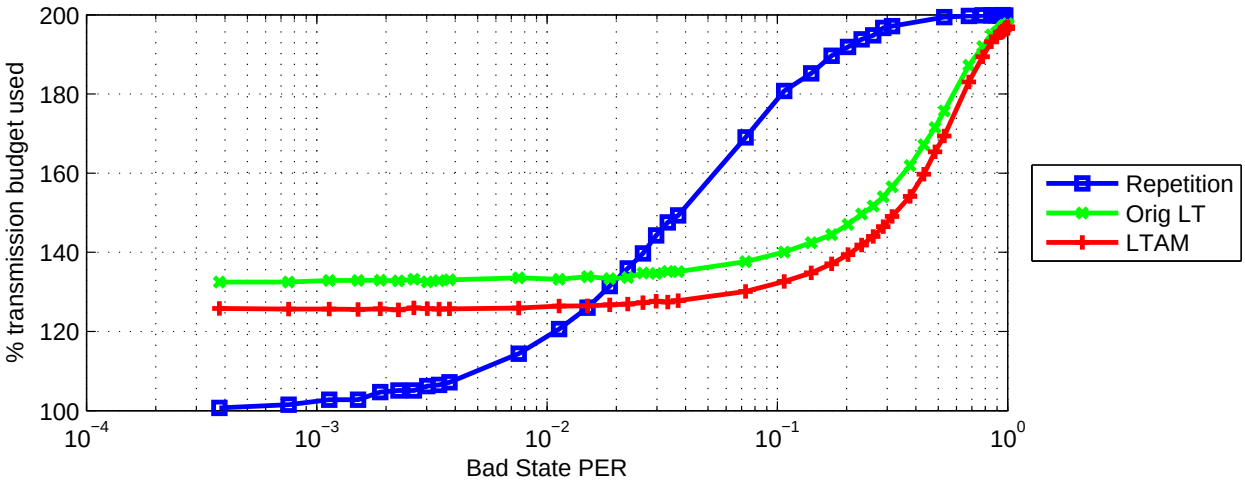

Figure 6.16: Transmission cost at different PER, GE channel, $R_c = 1/2$.

channels, and Figures 6.14, 6.16, 6.18 and 6.20 (pg.63, 64, 65 and 66) for burst error channels. This measurement, transmission cost, is presented as the average percentage of number of packets transmitted before the receiver can switch off normalised by k. It does not indicate whether the decoding is successful (this is indicated by the success rate measurement). This section has two subsections. Subsection 6.2.1 describes the results and Subsection 6.2.2 discusses the results.

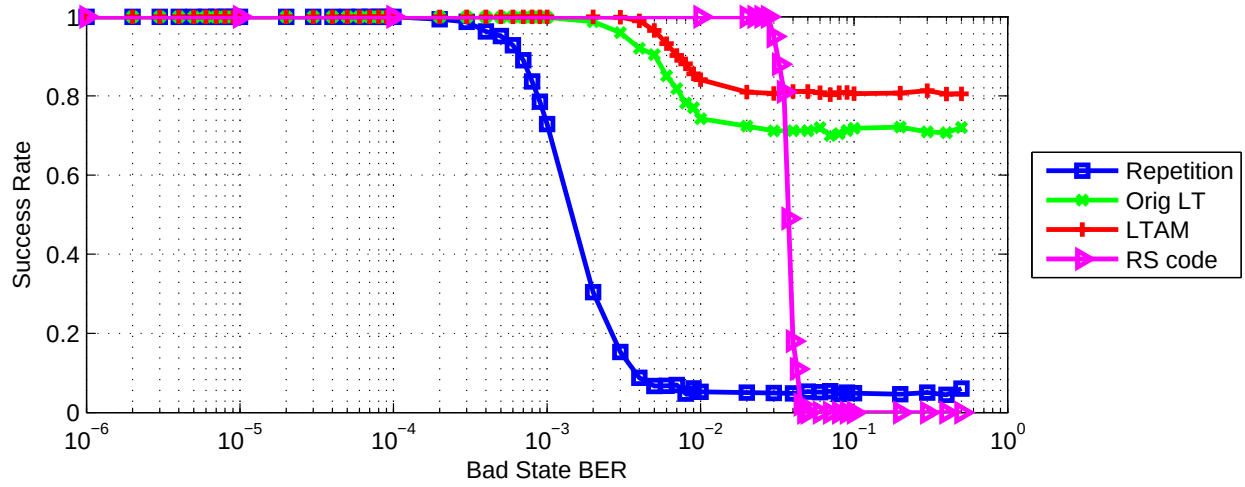

Figure 6.17: Success rate, GE channel, $R_c = 1/3$.

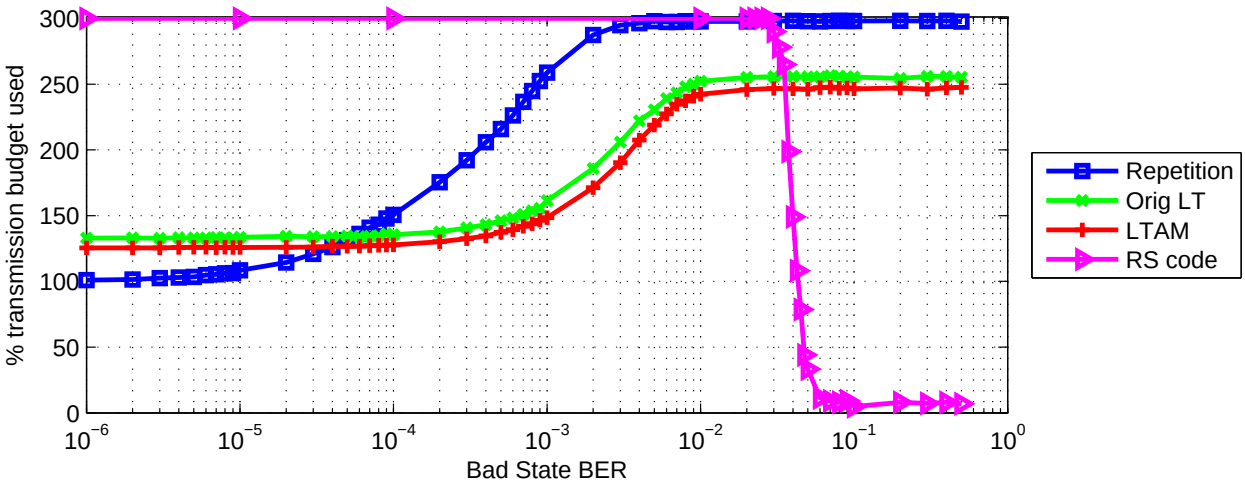

Figure 6.18: Transmission cost, GE channel, $R_c = 1/3$.

6.2.1 Transmission Cost Results

The transmission cost is a way of measuring energy consumption of data reception for the receivers by measuring the number of time slots that the receiver needs to stay active (effectively, the number of packets transmitted until decoding or end of transmission). This is not a measure of the number of packet received at the receiver because while a packet may not be successfully received, the time slot for receiving that packet still counts.

Although the description of the success rate and the transmission cost is outlined in separate

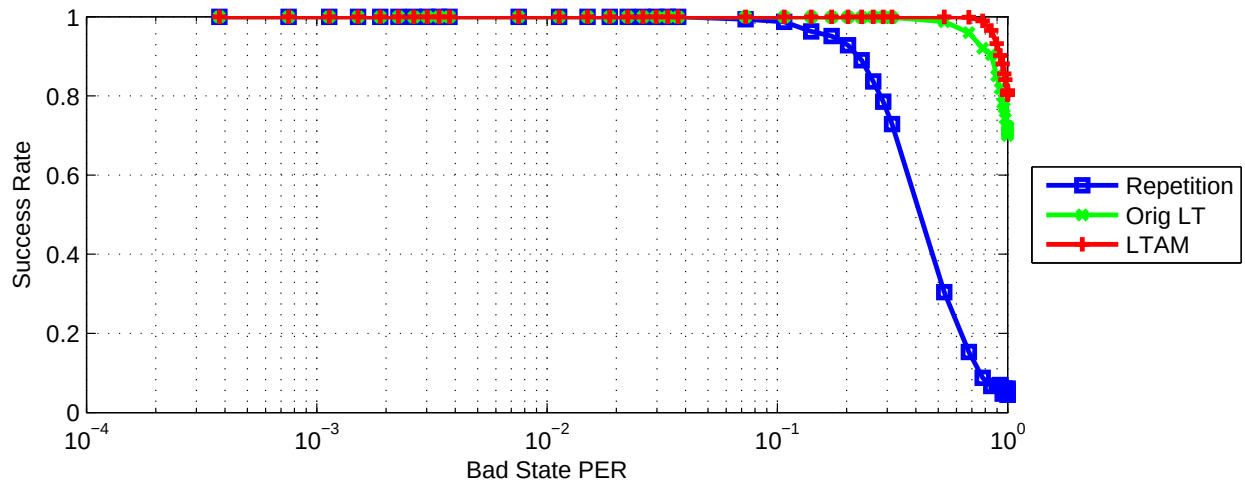

Figure 6.19: Success rate at different PER, GE channel, $R_c = 1/3$.

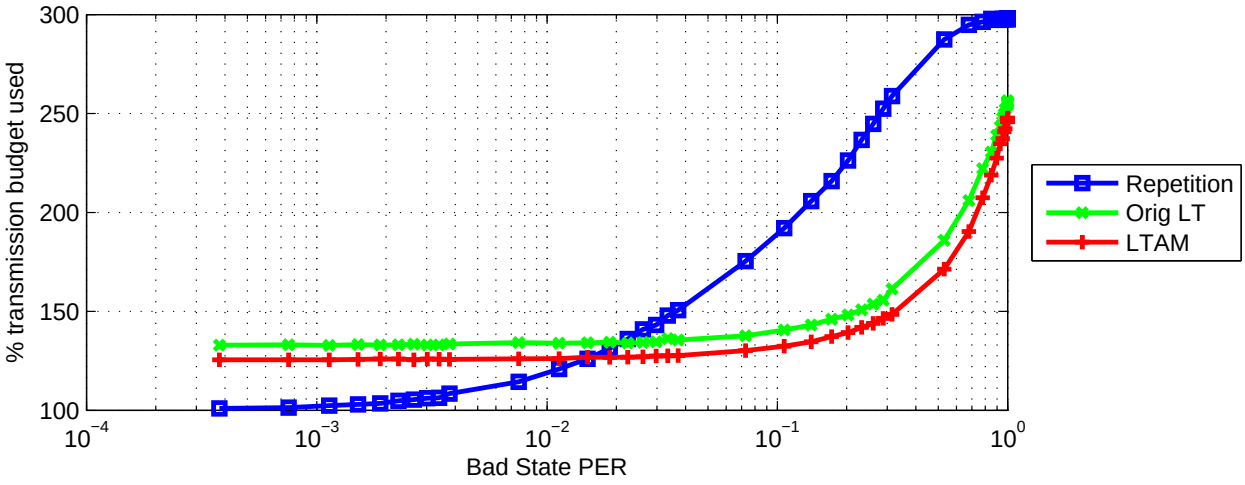

Figure 6.20: Transmission cost at different PER, GE channel, $R_c = 1/3$.

sections, the graphs for both measures for each channel model and code rate are placed close to each other for ease of comparison, since the curves for the transmission cost are directly related to those of the success rate measure. The relationships are:

- When the success rate is zero, the transmission cost is always maximum, i.e. 200% for $R_c = 1/2$ and 300% for $R_c = 1/3$.

- As the success rate decreases, the transmission cost increases.

- When the success rate is almost one, the transmission cost can vary depending on the mean channel error rate and the coding scheme used.

6.2.2 Transmission Cost Discussion

In this subsection, the transmission costs of the three main coding schemes (the RS codes, the LT codes and the repetition code) are discussed separately. Then the average percentages of correctly received packets of the LT codes and the repetition code are compared and discussed.

The RS Code

The RS codes always receives the total data transmission budget in order to successfully decode the user data since each encoded packet contains a unique part of the user data[6]. With a pre-defined code rate, the receiving process cannot finish early for successful transmissions even under channels with very low error rates as the pink line with triangles shows in Figures 6.2, 6.6, 6.14 and 6.18 (pg.53, 55, 63 and 65). However, the reception process can be immediately terminated after an encoded packet fails to decode in which case the entire transmission is assumed to be failed due to the permanent loss of some part of the user data contained in the lost packet. It can be seen from all eight figures (summarised in Table 6.1) that show the performance of the RS codes, the transmission cost decreases as the success rate of the RS codes decreases.

The LTAM Code and the Original LT Code

The LTAM code enables the receiver to switch off reception after about 126% of data is received when the PER is small (i.e. PER≤ 0.01) as shown as the red line with plus signs in Figures 6.4 and 6.8 (pg.54 and 56). The LTAM curve rises as the packet loss rate increases because more packets need to be transmitted with a higher channel loss rate before the receiver can collect enough packets for decoding. As the success rate starts to drop, the transmission cost still increases with a smooth trend. However, from the point that the success rate starts to drop, the increase in the average transmission cost is not only caused by increased number of transmitted packets for successful decoding, but also the cases of decoding failure that use the total transmission budget. As the

[6] Theoretically, a receiver can have a certain amount of switch off period during the reception of each packet when the channel is very good. Since the RS codes used are systematic, provided there is a checksum equipped with the information portion of each packet, the receiver can switch off for the rest of the packet reception time if it has successfully received the information portion of the packet. However it is not practical to implement such mechanism to real applications and therefore is not considered in this thesis.

success rate drops to zero at PER = 0.4 for $R_c = 1/2$ and PER=0.73 for $R_c = 1/3$, the total transmission budget is reached. The original LT code follows the same trend as the LTAM code. It requires more packets to decode than the LTAM code.

Figures 6.21, 6.22, 6.23, and 6.24 show the additional transmission cost of the original LT code over the LTAM code for the BEC channel with code rates 1/2 and 1/3, and the GE channel with code rates 1/2 and 1/3 respectively. As shown in Figure 6.21, for BEC channels and $R_c = 1/2$, the original LT code incurs on average 7.2% higher transmission cost than the LTAM code up to PER=0.01. From PER=0.01 to 0.2, the average difference increases slightly to 7.9%. From PER=0.2 to 0.5, the difference between them drops to zero, corresponding to the decreasing success rates which make the transmission cost for both schemes approach the maximum value. For $R_c = 1/3$ and BEC channels, as shown in Figure 6.22, the difference between the LTAM and the original LT code is 7.8% for $PER \leq 0.1$. The difference increases as the PER increases until the difference reaches a peak of 12.7% at PER=0.47. Then the difference decreases as the success rate decreases.

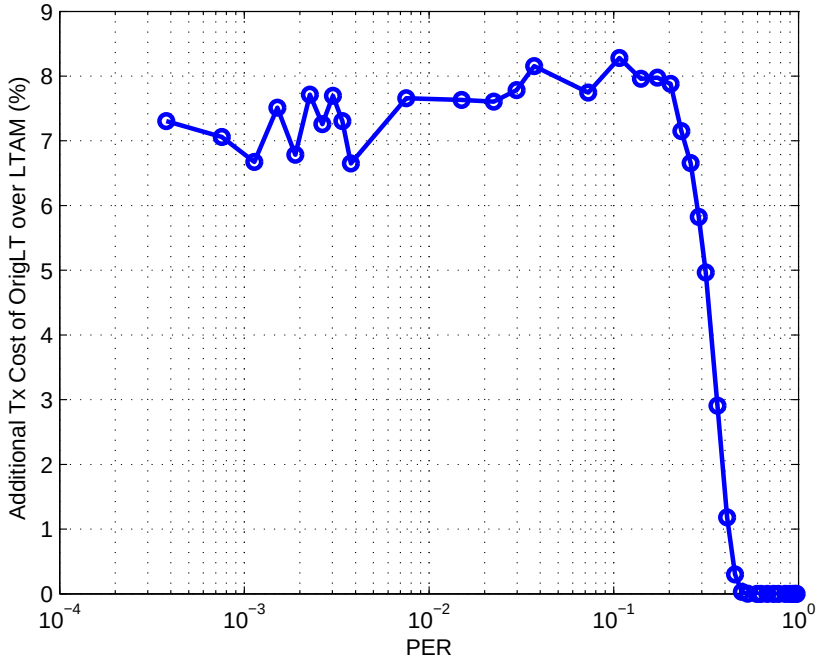

Figure 6.21: Additional transmission cost required for the original LT code over the LTAM code for BEC channels (statistical sample, size:5000), $R_c = 1/2$.

For GE channels, since the success rate is bounded away from zero for all PER_b tested, the

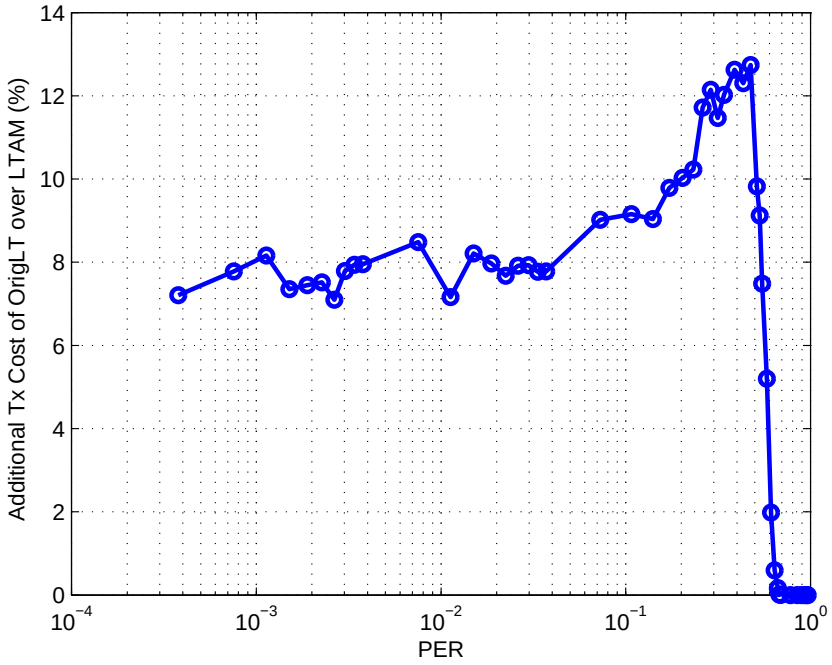

Figure 6.22: Additional transmission cost required for the original LT code over the LTAM code for BEC channels (statistical sample, size:5000), $R_c = 1/3$.

transmission cost never reaches the maximum for either LT scheme. The LTAM code and the original LT code level off at 197% and 198% respectively for $R_c = 1/2$ as shown in Figure 6.14 (pg.63) and at 248% and 255% respectively when $R_c = 1/3$ as shown in Figure 6.18 (pg.65). The LTAM code uses 7.2% less transmission cost than the original LT codes for PER less than 0.1 as shown in Figure 6.23 (pg.70). Similar to the BEC channel, there is a peak shown for $R_c = 1/3$ at about PER=0.7 before it decreases to zero. This again shows the advancement made by the LTAM code which enables early switch-off for the receivers.

The difference between the two schemes increases before dropping to zero and the peak is more pronounced for $R_c = 1/3$ than $R_c = 1/2$ for both channel models: compare the pair of Figures 6.21 and 6.22 (pg.68 and 69) and the pair of Figures 6.23 and 6.24 (pg.70 and 71). A range of code rates are tested for the difference in the transmission budget and it was found that as the code rate becomes lower, there is a higher peak in the difference between the two codes. This trend is explored more thoroughly in Section B.4 in Appendix B.

The trend for the peak transmission cost difference to increase for low code rates, such as $R_c = 1/3$ or lower, is because as the PER increases, the original LT code starts to have more cases of

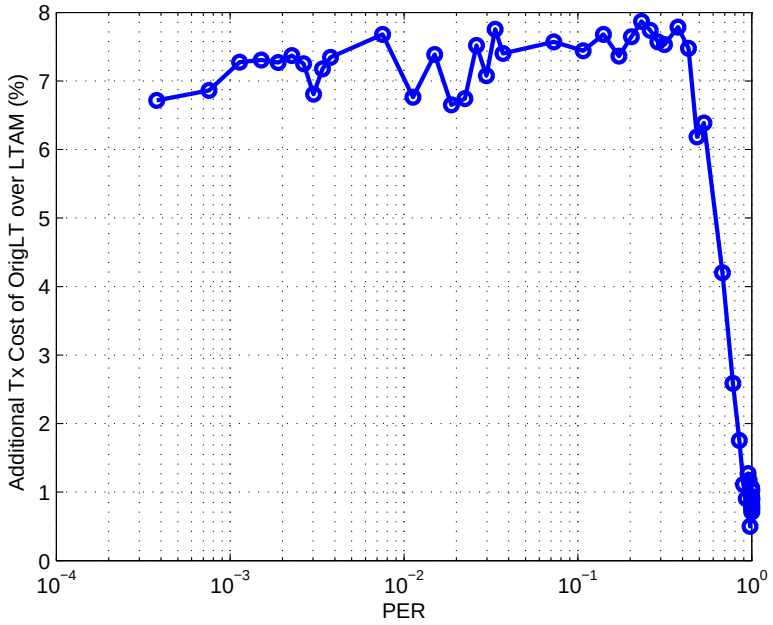

Figure 6.23: Additional transmission cost required for the original LT code over the LTAM code for GE channels (statistical sample, size:5000), $R_c = 1/2$.

decoding failure as the maximum transmission budget is reached, compared to the LTAM codes[7]. The transmission cost for both codes eventually reaches maximum when the PER increases, hence the difference drops to zero. For higher code rates such as $R_c = 1/2$, the transmission cost limit is reached earlier, before a significant upward trend is observed, so no clear peak is seen in Figures 6.21 and 6.23.

The Repetition Code

As the blue line with squares shown in Figures 6.4, 6.8, 6.16 and 6.20 (pg. 54, 56, 64 and 66), the repetition code uses the smallest budget to decode when the channel error rate is small. It is expected because the repetition code used was a systematic code (the REPB sequence was chosen). Systematic codes are the most efficient coding scheme under no loss channels or channels with very small error rates.

For the memoryless channel, the repetition code uses only 101.8% of the total budget when PER=4×10^{-4} for both code rates. The repetition code has the least reception budget for the

[7] This is due to the heavier tail of the decoding distribution shown in Figures 6.10 and 6.11 (pg.59 and 60)

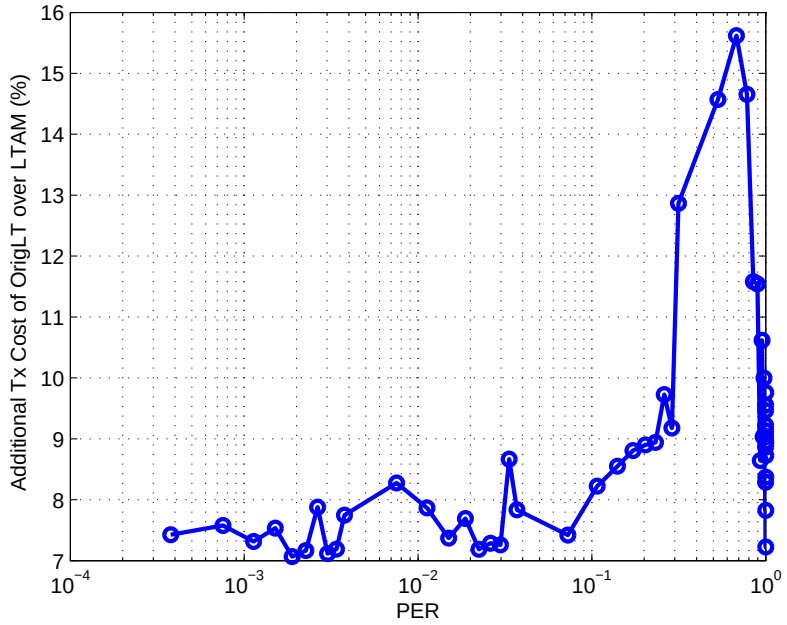

Figure 6.24: Additional transmission cost required for the original LT code over the LTAM code for GE channels (statistical sample, size:5000), $R_c = 1/3$.

memoryless channel up to PER=0.007 (for both code rates) where the cross-over point occurs between the repetition code and the LTAM code. For burst error channels, the cross-over point between the repetition code and the LTAM code occurs at $PER_b = 0.015$ for both code rates. For both channel models the cross-over point occurs when the transmission cost is 126.5%. As the success rate is bounded away from zero in the case of high PER_b of the GE channel and $R_c = 1/3$ for the repetition code, the maximum transmission cost for the repetition code is 297%. Although repetition code with REPB sequence is shown to have least error tolerance, it is the most energy saving scheme when the channel error rate is very small.

Number of Packets Needed to be Collected at the Receiver for Decoding

Figures 6.25 and 6.26 show the percentage of total number of packets correctly received normalised by k (shortened as budget received) at the receiver before the receiver can switch off for the three packet erasure coding schemes for $R_c = 1/2$ and $1/3$, respectively. It can be seen that both the LTAM and the original LT code need consistent numbers of packets for successful decoding. The curves start to decrease due to limited transmission budget and higher channel loss rate. A graph

that specifically shows the budget received difference between the original LT code and the LTAM code is presented Figure B.15 in Appendix B.

The budget received for the repetition code increases as the channel error rate increases up to a point as shown in Figures 6.25 and 6.26. It starts decreasing again due to limited transmission budget and higher channel error rate. For $R_c = 1/2$, the peak occurs at PER=0.07 and a maximum of 176% of budget received. For $R_c = 1/3$, the peak occurs at PER=0.2 and it has a maximum of 226% of budget received.

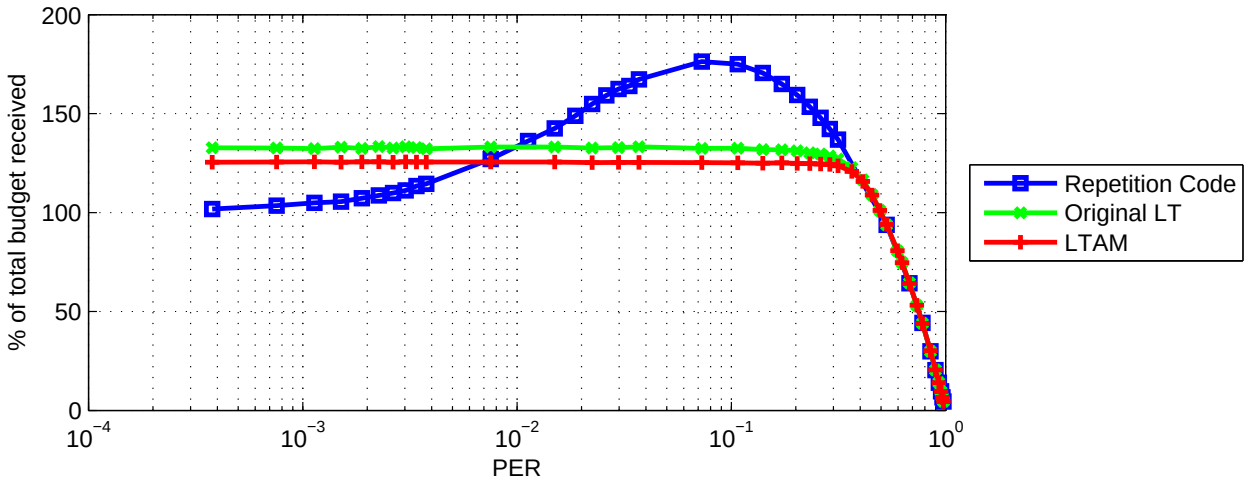

Figure 6.25: Budget (correctly) received at different PER, memoryless channel, $R_c = 1/2$.

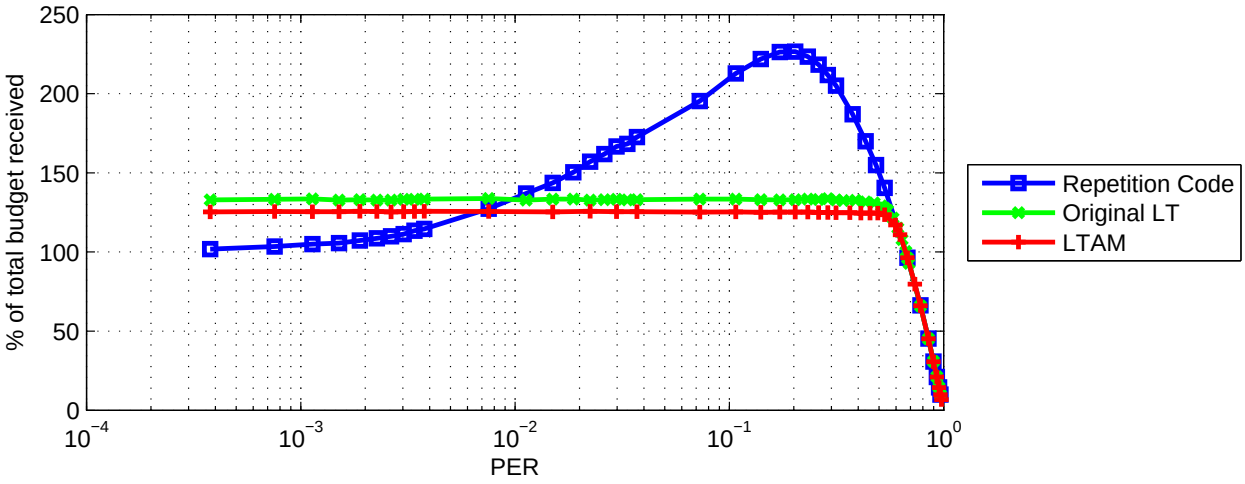

Figure 6.26: Budget (correctly) received at different PER, memoryless channel, $R_c = 1/3$.

6.3 LT codes combined with RS codes

From the success rate graphs shown in the previous two sections, it can be seen that there is a big performance degradation from the RS codes to the packet erasure coding schemes in terms of error tolerance. This is because there is no mechanism in the packet level erasure codes to correct errors within a packet; only a correctly received packet enters the decoding process of the erasure coding schemes. The correctness of a packet is limited by the channel error rate. As introduced by Elias [10], the maximum capacity of an erasure channel is $1 - PER$. If there are k packets of user data, there needs to be at least the same number of packets (with the same size) successfully arriving at the receiver in order for successful data transmission. This limits the performance of the packet erasure codes. The dotted cyan line in Figure 6.27 gives an indication of the capacity that can be achieved by packet level erasure schemes with packets of size 378 bits with no intra-packet error correction technique. The line shows the average probability of receiving any k packets and each packet contains 378 bits (this packet size is used for the simulations). Figure 6.27 shows that LT codes are very close to the capacity of the erasure channel and the barrier in keeping high success rate at higher channel error rate is the unprotected LT encoded packets that are too vulnerable to channel noise.

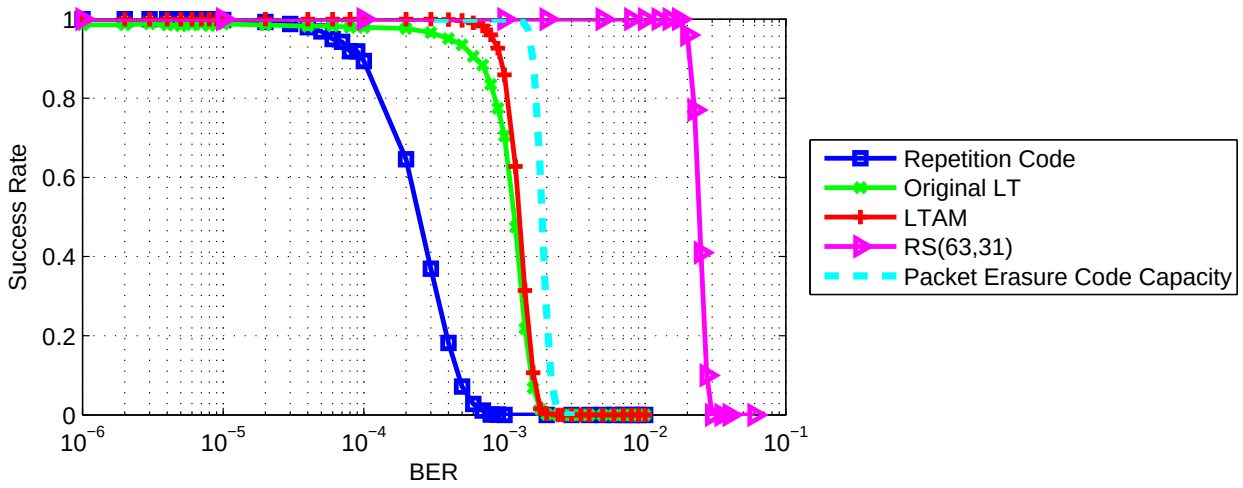

Figure 6.27: Success rate, memoryless channel, $R_c = 1/2$.

In order to utilise the good features of the LT codes (i.e. the ability of keeping a high success probability over long burst channels, and switch off receivers with good channels early) together with the good features of the RS code (i.e. strong error correction capability), it was decided that an

RS code be concatenated to the LT code to improve the success probability of receiving individual LT packets. This newly proposed scheme is called LT&RS codes.

Figure 6.28 shows the block diagram of the simulation model for the LT&RS codes used. The LTAM code is used in this scheme as it has a better performance than the original LT code. The packetised user data are first encoded into LTAM encoded packets. The LTAM encoded packets are then encoded into RS(63,55) encoded packets. At the decoder, the received data first goes through an RS(63,55) decoder and the successfully decoded packets enter the LT decoder.

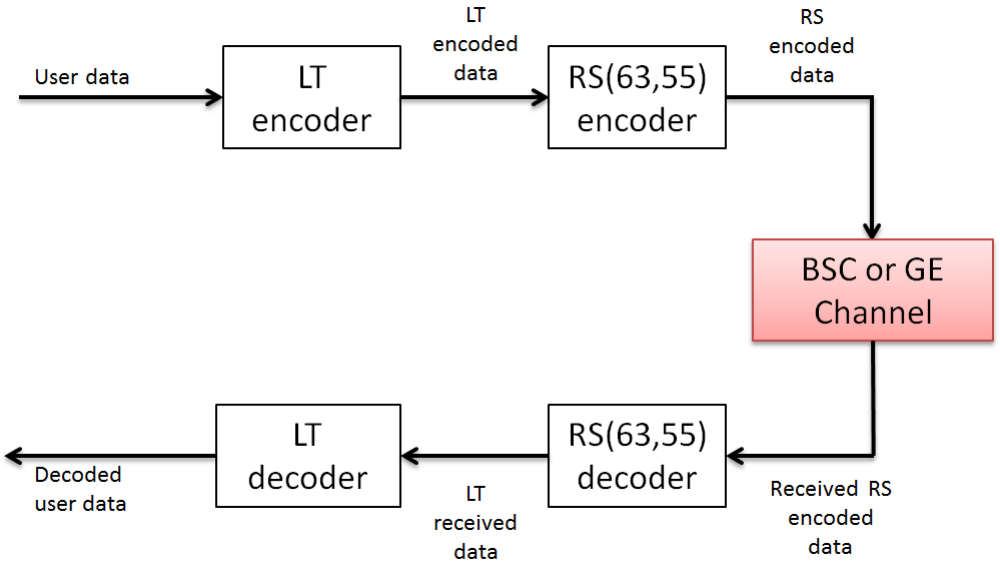

Figure 6.28: A block diagram of the transmission model for simulating the LT&RS codes.

6.3.1 LT&RS Code Structure

Three factors of the LT&RS code need to be kept consistent with the other coding schemes[8] in order to make valid comparisons between this scheme and the others. They are the packet size, user data size and code rate. The RS(63,55) code has chosen to be used in this scheme. n_{RS} is chosen to be 63 in order to keep the packet size consistent. k_{RS} is chosen to be relatively high so that the LT code dominate out and hence most available budget should be dedicated to constructing LT encoded packets. The RS code is added to give the packet a limited degree of protection to improve the overall efficiency of the LT code. The amount of user data is again set to 32384 bits

[8] For packet structure details of the other coding schemes, refer to Tables 2.1 and 2.2.

in order to keep the user data size consistent. Finally, to keep the overall budget consistent, the following calculation is performed:

Given that each encoded packet contains 330 user data bits (k_{RS}=55×6), there needs to be 99 packets to accommodate all 32384-bit ($32384/330 = 98.133$) user data. This means there are 99 LT encoded packets assuming that the LT encoded packet has the same size as the user packet. A certain amount of budget is used for RS encoding (63-55=8 symbols or 48 bits per encoded packet which is 4752 bits (48×99) for all 99 LT encoded packets). For $R_c = 1/2$, the total budget is 64768 bits (32384×2), 99 LT&RS encoded packets used up 37422 bits. Therefore only 73 extra LT&RS encoded packets ($(64768 - 37422)/378$) can be generated to give a total of **172 encoded packets**. For $R_c = 1/3$, the total budget is 97152 (32384×3), therefore 159 extra encoded packets can be formed which makes a total of **258 encoded packets**.

Since the user data is split into more packets in this scheme (each packet contains less user information), the distribution shown in Figure 6.11 (pg.60) will shift to the right, including the tail of the distribution. It was expected that this will affect the success probability of the code under no-loss channels as well as lossy channels especially for code rate 1/2. However, as shown in Figure 6.29, the effect in success probability by increasing the user packets from 86 to 99 is very small. In the 10,000 experiments executed under loss-free channels, there were four cases in the LTAM scheme that exceeded the $2 \cdot k$ budget and 11 cases in the LT&RS scheme which exceeded the $2 \cdot k$ budget contributing 0.11% of the number of experiments instead of 0.04% in the case of the LTAM scheme.

6.3.2 LT&RS Result Description

Table 6.4 provides a summary of the figure numbers and their page numbers grouped according to their corresponding channel models and code rates, again for quick reference. The performance curves of the four initially proposed schemes shown in the graphs are exactly the same curves that are presented in the previous two sections. In this section, the LT&RS performance curves are added in the graphs that are listed in Table 6.4.

The LT&RS code shows a significant improvement in success rate over the LTAM codes. For both error channel models, the ability to achieve nearly 100% success rate over a range of BERs by as much as about one order of magnitude as shown in Figures 6.30 and 6.32 (pg.79 and 80)for BSC channels and Figures 6.34 and 6.36 (pg.81 and 82)for GE channels. The improvement is due to a much higher single packet reception rate (the probability that a packet is decoded successfully by the RS decoder), which becomes the input to the LTAM decoder.

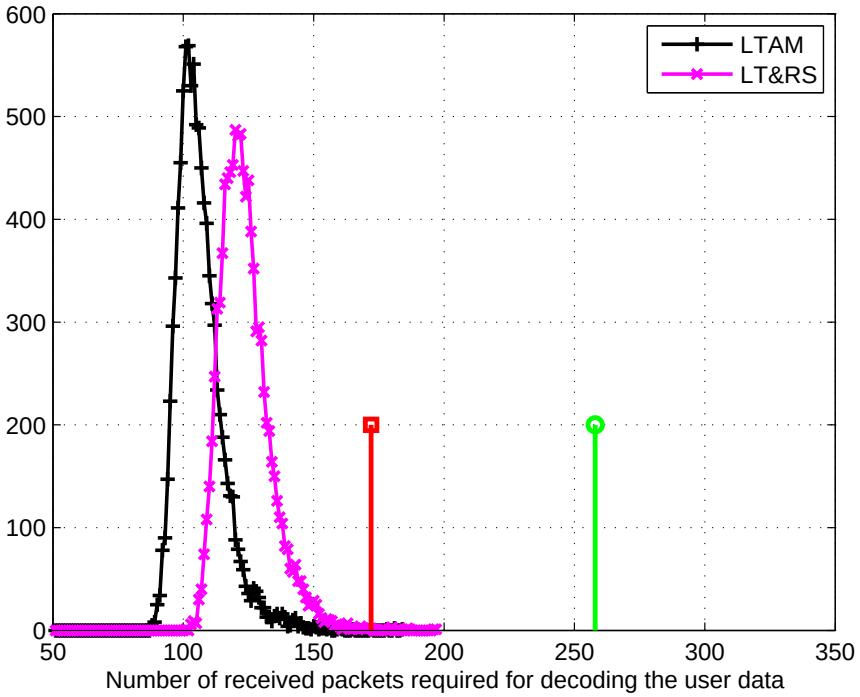

Figure 6.29: The distributions of the numbers of received packets needed for decoding the LT&RS and LTAM code over loss-free channels. The red bar with a square tip indicates $2 \cdot k$ budget and the green bar with a circle tip indicates $3 \cdot k$ budget.

From Figures 6.31 and 6.33 (pg.79 and 80), it can be seen that for BER$< 1 \times 10^{-5}$, the transmission cost is 7% higher than for the original LT codes and 16% higher than the LTAM code due to the fact that a slightly larger k is needed for the LT decoding part ($k = 99$ rather than 86). More budget is dedicated to the RS coding causing less information per packet. However, the transmission cost for the LT&RS code outperforms the LTAM code from BER=3×10^{-4} for the BSC channel and $BER_b = 5 \times 10^{-4}$ for the GE channel due to a higher success rate which enables less transmission cost.

The key observation for the LT&RS coding scheme is that it shows a significant improvement over LT codes for the success rate while the increase in total budget used is relatively small. It is considered to be the best scheme within this project in satisfying the aim – maximising the success rate while keep the energy consumption in data reception low.

Table 6.4: A summary of figure numbers that show the performance comparison for the four proposed coding schemes and LT&RS scheme.

Channel	Code Rate	Measure	Figure	Page Number
BEC/BSC	$1/2$	Success Rate	6.30	79
		Budget Used	6.31	79
	$1/3$	Success Rate	6.32	80
		Budget Used	6.33	80
GE	$1/2$	Success Rate	6.34	81
		Budget Used	6.35	81
	$1/3$	Success Rate	6.36	82
		Budget Used	6.37	82

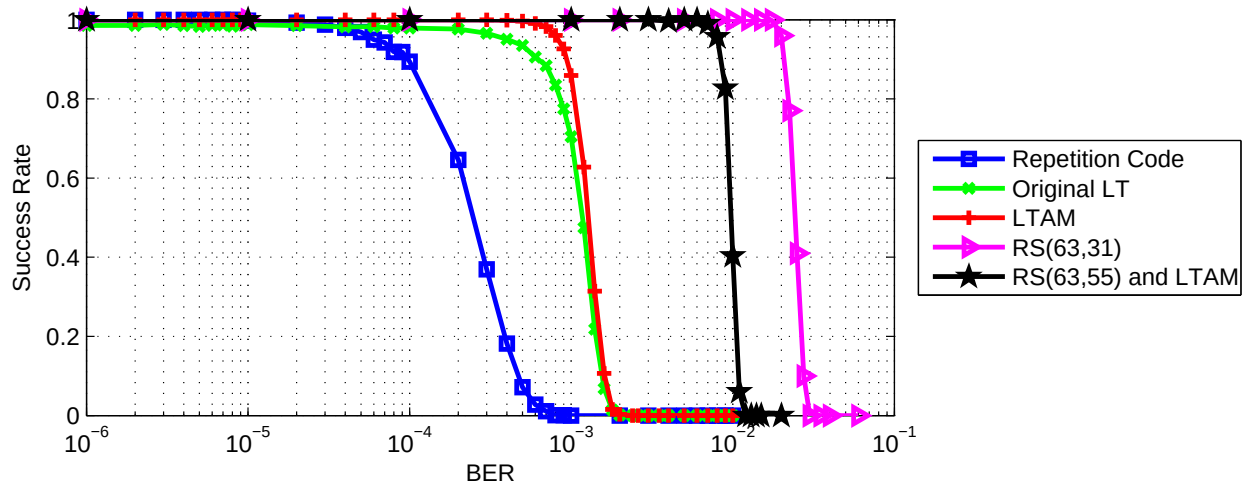

Figure 6.30: Success rate with LT&RS code added, memory channel, $R_c = 1/2$.

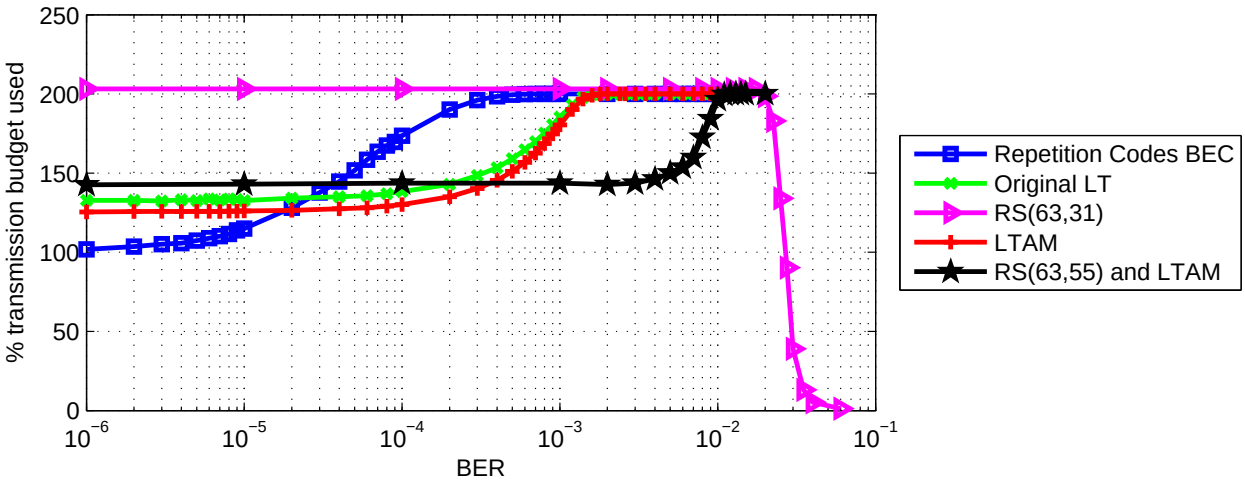

Figure 6.31: Transmission cost with LT&RS code added, memoryless channel, $R_c = 1/2$.

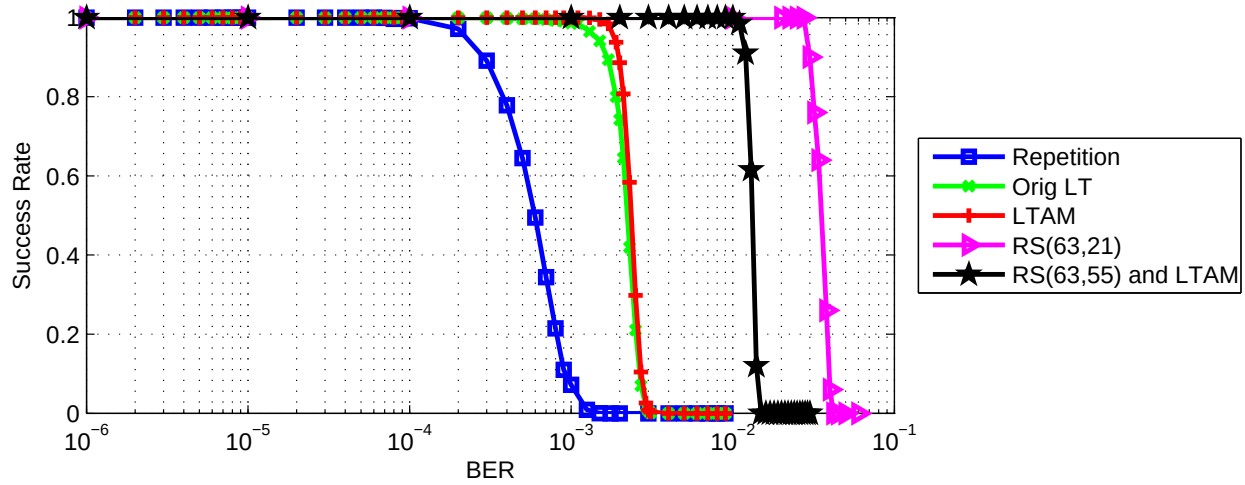

Figure 6.32: Success rate with LT&RS code added, memoryless channel, $R_c = 1/3$.

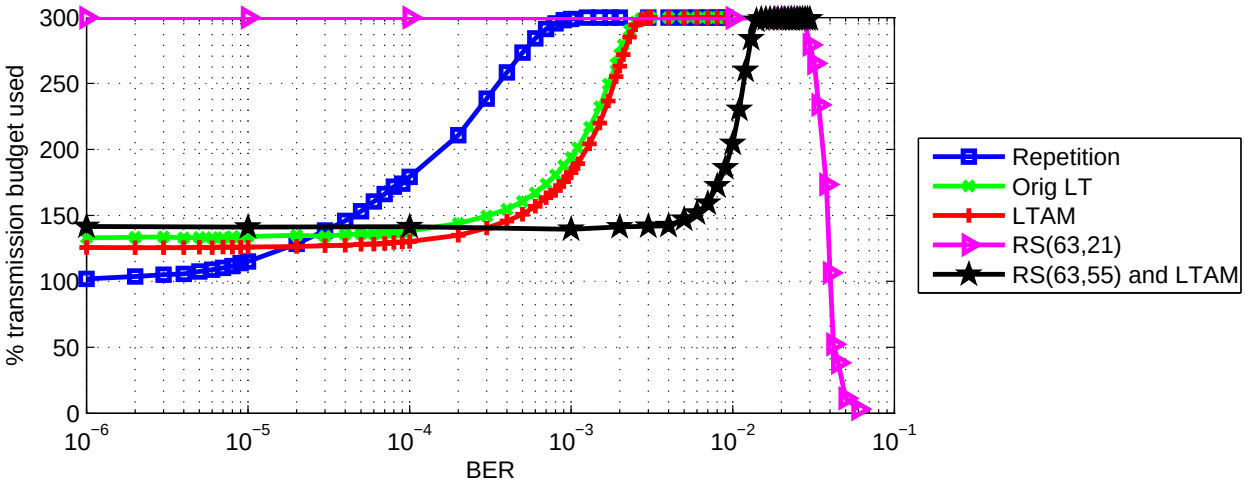

Figure 6.33: Transmission cost with LT&RS code added, memoryless channel, $R_c = 1/3$.

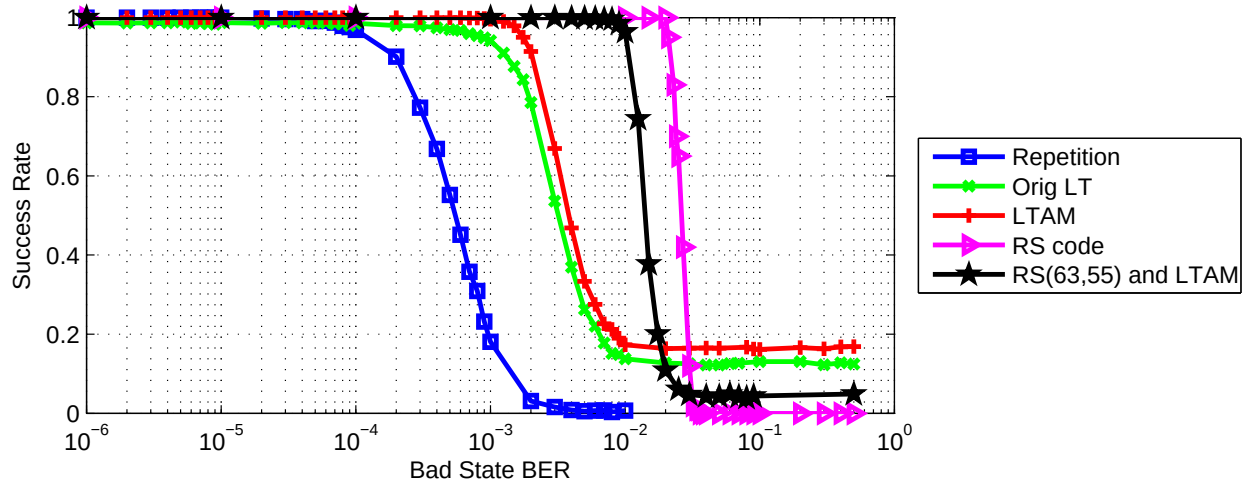

Figure 6.34: Success rate with RS&LTAM code added, GE channel, $R_c = 1/2$.

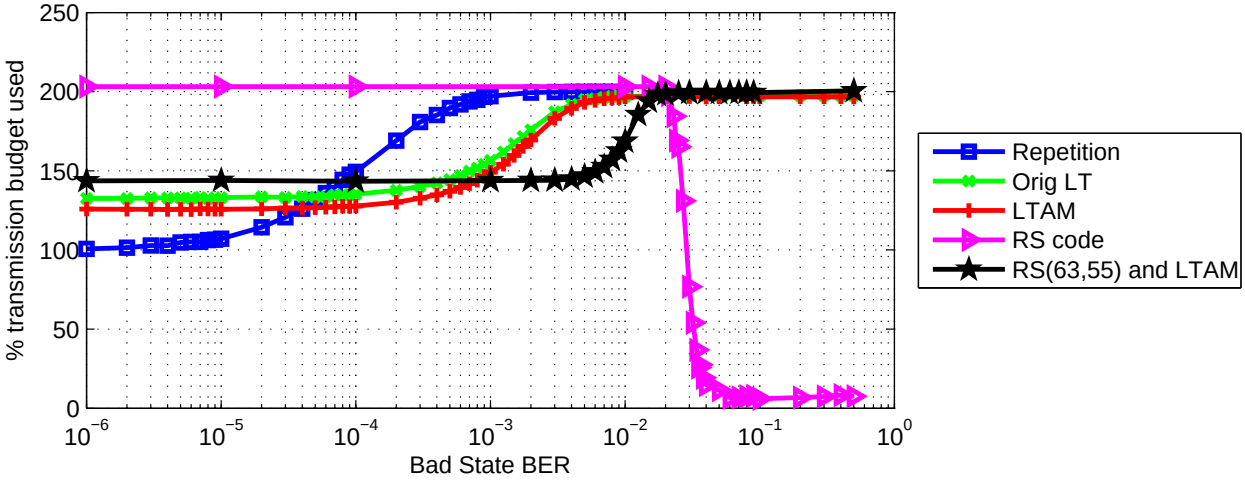

Figure 6.35: Transmission cost with RS&LTAM method added, GE channel, $R_c = 1/2$.

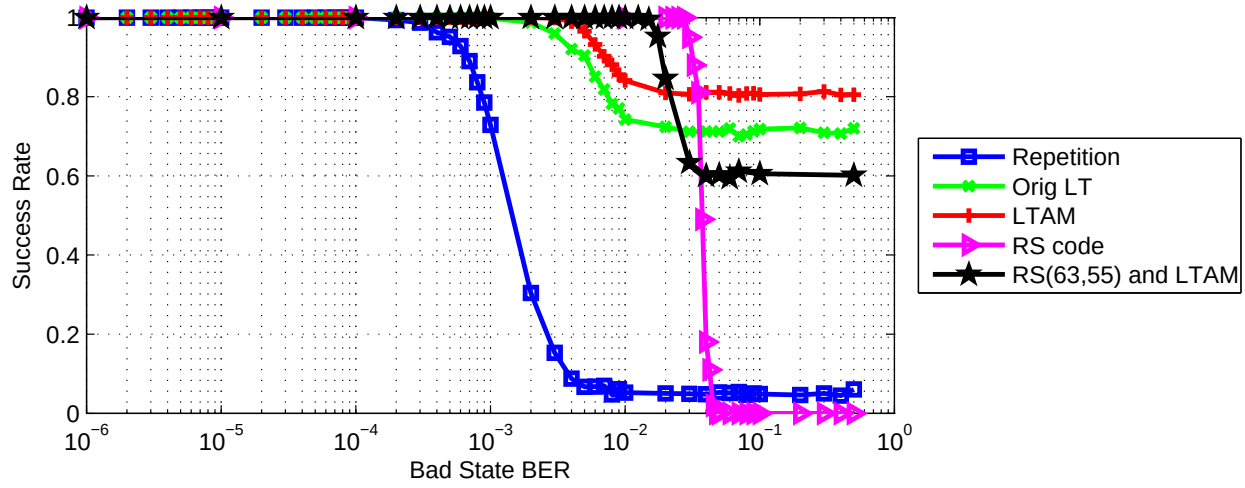

Figure 6.36: Success rate with RS&LTAM method added, GE channel $R_c = 1/3$.

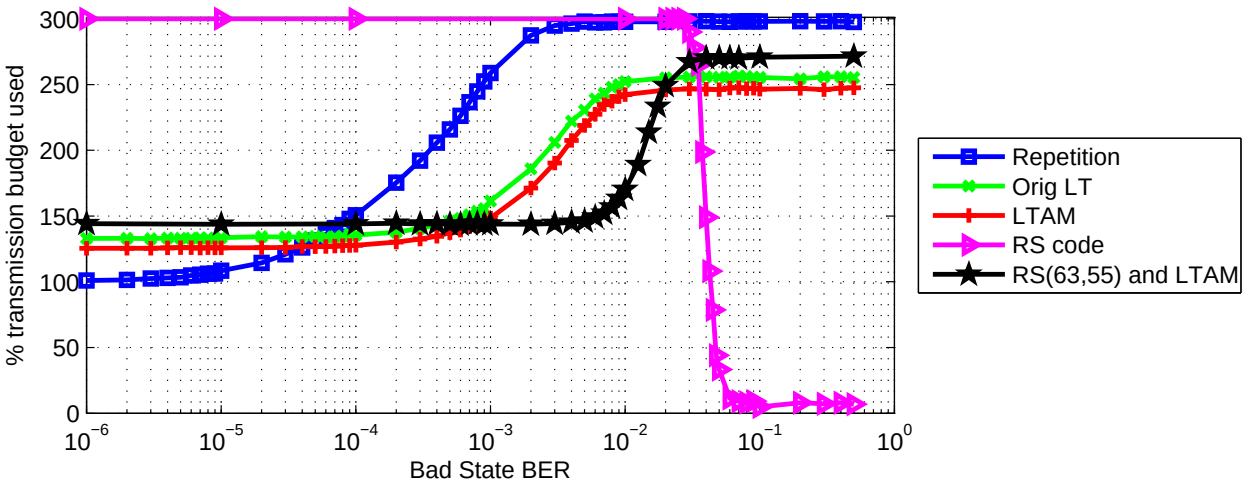

Figure 6.37: Transmission cost with RS&LTAM method added, GE channel, $R_c = 1/3$.

6.4 Comparison of the Complexity of the Candidate Codes

Although the decoding complexity of the codes is not one of the main considerations, it affects the energy consumption for decoding the data at the receiver and therefore influences the choice of the most energy saving scheme for the application communication system. The decoding complexities or cost of the coding schemes are briefly described in this section.

The decoding costs of the coding schemes are shown in Table 6.5. Repetition codes have an $O(1)$ decoding cost since there is no real decoding operations needed for this scheme; all the decoding process does is check the checksum (common to all coding schemes). RS codes have a per packet decoding cost in the order of $k_{RS}(n_{RS} - k_{RS})log_2(n_{RS})$ [5, 28] where k_{RS} and n_{RS} are the number of user symbols and total number of symbols per RS encoded packet respectively. This is a measure of the number of `XOR` operations needed for decoding an RS code. Detailed RS decoding using `XOR` operations can be found in [3]. The complexity of decoding RS codes has a quadratic relationship with the packet size and since each packet is constructed independently, the relationship between the decoding complexity and the total number of packets is linear. Therefore the complexity of decoding RS codes can be expressed as $O(j^2 \cdot n)$ where j is the number of bits per encoded packet and n is the number of encoded packets. The decoding complexity of the LT codes is $O(j \cdot n \cdot ln(n))$ as suggested by [5]. The complexity of the LT&RS code is therefore $O(j^3 \cdot n \cdot ln(n))$. In summary, repetition codes have the least decoding cost, LT codes have lower decoding cost than RS codes and LT&RS codes have the highest decoding cost due to the fact they combine the decoding costs of LT codes and RS codes.

These are very general indications of the decoding costs of the schemes. Specific energy consumption for decoding depends on many factors. These include the choice of the decoding algorithms used and the efficiency of the actual software/hardware realisation and implementation.

Table 6.5: Decoding costs of the candidate coding schemes.

Code Name	Decoding Complexity
Repetition codes	$O(1)$
RS codes	$O(j^2 \cdot n)$
LT codes	$O(j \cdot n \cdot ln(n))$
LT&RS codes	$O(j^3 \cdot n \cdot ln(n))$

6.5 Conclusion

This chapter presented a performance comparison of five coding schemes: the repetition code, the RS code, the original LT code, the LTAM code and the LT&RS code. RS codes are the scheme that can keep a close to 100% success rate under worst memoryless channel conditions. However, RS codes do not allow early switch off for the receivers due to the fixed code rate and therefore is not particularly suitable for broadcast/multicast systems where the channel variations between the transmitter and different receivers are large. The systematic repetition code can achieve the lowest transmission cost when the channel loss rate is very small; however, it has a poor error tolerance ability. The LT codes are the coding scheme that both have significant improvement to the success rate over the repetition code and are able to switch off early for receivers with good channels. Moreover, the LT codes achieve a higher success rate than the RS codes in burst error channels where the BER_b is high. By combining LT and RS codes together, a clost to 100% success rate can be achieved over channels with even higher error rate, it is very close to the performance of the RS codes tested. At the same time, an early switch off is possible for receivers with small error rate channels with the LT&RS code. This coding scheme – LT&RS is the best coding scheme tested that fulfils the initial aim of this project.

Chapter VII

Related Work and Future Work

This chapter introduces the related work to this thesis. Section 7.1 describes related work specifically to the LT codes, Section 7.2 describes related work on the general field of broadcast communications and Section 7.3 discusses possible future work.

7.1 Related Work to LT Codes

Several papers have focused on improving the decoding efficiency of LT codes for small message sizes. Nearly all of these improve the degree distribution of LT codes. In [7] a Kent Chaotic map is used to draw random variates from the Robust Soliton distribution. Their overhead for $k = 2000$ is 7.3% whereas the overhead of the original LT code 9.1% overhead and the LTAM code has an overhead of 7.12%. In [53], a suboptimal degree distribution is proposed for LT codes after the authors identified some drawbacks of using the optimal degree distribution in practice. Results for this scheme are shown and discussed in Section 5.4. The LTAM scheme shows better performance over the results shown in [53] for k less than 1000 by about 2%. In [52], evolution strategies are used for finding degree distributions of optimal short length LT codes ($k < 10^3$) for different applications. Reference [22] improves the decoding process by introducing a decoding algorithm called full rank decoding which uses the Wiedemann algorithm. Since the performance measures used in this thesis are different to the aforementioned two schemes, the results are not directly comparable. In [17] two degree optimization approaches are taken: a Markov chain approach and a combination method. However, their methods are not scalable for $k > 30$.

7.2 Related Work on Error Control for Broadcast Systems

This section describes other work that have been done in the field of error control for broadcast systems. The first subsection discusses systems with only forward channels and the second subsection describes systems with both forward and feedback channels. Although feedback channels

are not applicable to the work of this thesis, some related work that involves feedback channels are included for completeness.

7.2.1 Systems with Only Forward Channels

In [19], an improved FEC scheme for 3D HDTV is proposed. The proposed scheme concatenates the Block Turbo Codes (BTC) with LDPC codes and compares the performance with the FEC scheme used in the current DVB-T2 standard which uses BCH codes combined with LDPC codes[9]. The proposed scheme has shown to have better BER performance under the Rayleigh slow fading channel conditions at high SNRs ($E_b/N_0 > 4.5dB$) where the BCH and LDPC combination starts to have an error floor but the BTC and LDPC combination is able to decrease the BER for higher SNR. The FER (Frame Error Rate) performance of the BTC and LDPC has also shown to be better than that of the BCH and LDPC because the performance of BTC is not limited by the number of error corrections like in the case of the BCH code.

A feature that is used in DVB-H (Digital Video Broadcast-Handheld) to combat short burst errors is called time slicing. In DVB-H systems, it is used along with the FEC scheme called MPE-FEC (Multi Protocol Encapsulation - Forward Error Correction) which employs the RS(255,191) code. Time slicing is a technique of sending block of data in bursts. Time slicing together with MPE-FEC not only provide a better protection of the transmitting data for impulse interference [13, 35] but also against Doppler effects in an urban environment [16, 38].

In DVB-SH (Digital Video Broadcast - Satellite services to Handhelds), the error control technique is improved to MPE-iFEC (MPE-inter-burst FEC) which initially uses Sliding Reed Solomon Encoding (SRSE) [16] and later supports Raptor codes [38]. MPE-iFEC is an improvement of MPE-iFEC in a respect that MPE-iFEC is able to repair long burst errors that happens often in satellite communications whereas with MPE-FEC, if a block of data is completely lost, it can no longer be recovered. The ability of repairing long burst errors comes from the interleaving property of the SRSE [15]. For Raptor codes (as a type of fountain codes), its data recovery ability is not affected by the pattern of the data losses.

In [45], the authors compared packet level RS codes and repetition codes in terms of their efficiency in DVB transmissions using IP multicast over Wireless LANs. Their results shown that the repetition code is always more efficient when the packet loss ratio (PLR) is small, otherwise RS codes perform better. This is consistent with the findings of this thesis.

7.2.2 Systems with Both Forward and Feedback Channels

Reference [2] finds efficient error control method for an HDTV IP broadcast scenario. Both FEC and ARQ are exploited. The technique that the authors proposed in this paper is: error correcting packets are split into two sessions, fundamental and supplementary. In the fundamental session the transmitter transmits data to all receivers. This session contains all user data as well as a small amount of error correcting data. Thus, this session has only minimal protection from channel errors, but it helps receivers with good channels to finish reception early. The supplementary session contains additional error correcting information divided into blocks and it is for receivers with higher channel losses that did not successfully recover the user data during the fundamental session. When the transmitter receives requests for retransmission from the ARQ process, it transmits the corresponding block during the supplementary session. The transmitter sends the same supplementary block in one go if several receivers request the same data block within a time window. This reduces wasteful transmissions.

The error control technique used in [20] is also a combination of ARQ and FEC. This paper focused on multicasting IPTV in Wireless LAN. The main analysis is on different ARQ protocols, namely, the Leader Based Protocol (LBP), Beacon-based LBP (BLBP) and Hybrid LBP (HLBP). FEC is used in the HLBP only as a general technique for increasing the reliability of packet reception. The working principle of LBP is that there is a leader elected from the receivers in a multicast system. Only the leader sends ACK[1] and the rest of the receivers only send NACK[2]. In the case of successful transmission for all receivers, the transmitter receives an ACK from the leader. If the transmitter does not receive an ACK, it may mean that the ACK from the leader has collided with NACKs sent from other receiver(s). The transmitter then retransmits the data until the maximum number of attempts is reached. The example FEC technique used in the HLBP is RS codes.

7.3 Future Work

7.3.1 Raptor Codes

Raptor (Rapid tornado) codes [41, 42] are an extension to LT codes. Raptor codes achieves linear encoding and decoding time, and are very close to ideal code performance under any channel loss patterns [24, 41, 6]. Raptor codes have been standardized as the application layer FEC for the 3GPP MBMS (3rd Generation Partnership Project Multimedia Broadcast Multicast Services)

[1] ACK: Positive acknowledgement of a packet reception.

[2] NACK: Negative acknowledgement of a packet reception.

cellular network [1]. They are also adapted in the DVB-H systems [12] for file delivery applications. Raptor codes can be non-systematic or systematic. As to whether the Raptor codes have better performance in terms of success rate and transmission cost than the RS&LT codes, further investigation is needed.

7.3.2 LTAM Codes

There is a possibility that the LTAM code can be further improved by using a better degree distribution than the current Robust Soliton distribution, and/or adding more encoding rules to the existing four cases. This can be further investigated in the future. In the case of unlimited transmission budget, a window can be applied to the LTAM code for storing the history encoding information for generating new encoded packets in order to keep the memory needed at the transmitter determinable.

7.3.3 Comparison between the Two LT Decoding Processes

A slightly different LT decoding process is used in this project than that described in [23]. A new decoding possibility was identified: two received packets with degrees d and $d+1$, which sharing d common neighbours. In this situation the extra user packet contained in the received packet with degree $d+1$ can be decoded. This decoding method is suspected to have better decoding efficiency. Further investigation is needed to prove this.

7.3.4 A More Specific Channel Set-up

In this thesis, only BSC channels and GE channels with symmetrical transition probabilities were used for the simulation. If the real channel statistics of the focused application can be obtained in the future, a more realistic GE channel parameters could be used. Furthermore, a more general Markov chain model could potentially model the fading channel characteristics more accurately.

Chapter VIII

Conclusions

The aim of this project was to find an FEC scheme that can achieve both high average success probability and low energy consumption of data reception for the receivers in a wireless broadcast system where the total transmission budget is fixed. Three candidate coding schemes were initially proposed – repetition codes, RS codes and LT codes. The coding schemes were applied at the data-link layer and the investigations were carried out by MATLAB simulations. The two performance measures were success probability and time until the receiver switches off represented as the transmission cost. The system model defines a set of simulation parameters. These are the user data length, code rates and channel types. Simulation models were built for the proposed coding schemes and the performance of the simulation models was tested and validated.

During the course of the investigation of the LT codes, an improvement to the original LT encoding process was made, it was given the name the LTAM codes. The LTAM codes reduce the decoding overhead for small message lengths. The improvement can be seen for messages of up to 10,000 packets. The overhead reduction is as much as 10% compared to the original LT codes.

The LT type codes were found to be superior in that they could achieve near erasure channel capacity performance and support flexible switch off time at the receivers. They are also adaptable to different channel characteristics; high success rate can be maintained over long burst error channels as long as enough number of packets reaches the receiver. Therefore it is a prototype of the ideal coding scheme that this project was seeking. This scheme was then further developed by applying an RS code as an inner code to the LTAM code to further improve the success probability of packet reception. The result shows that the LT&RS code has a significant improvement in the ability of channel error tolerance with only slightly more transmission cost over that of the LT code without an RS code applied. This LT&RS code is then determined to be the best scheme that fulfils the aim.

Comparing the LT&RS codes with the baseline repetition codes, the improvement in success rate is two orders of magnitude: the LT&RS code can keep full success rate over channels having approximately 100 times (two orders of magnitude) more errors compared to the repetition code.

While the success rates are improved, the transmission cost can be maintained as low as 142% up to the knee of the success rate. This is 58% reduction in data reception for $R_c = 1/2$ and 158% reduction for $R_c = 1/3$ over the base line repetition code as well as the RS code.

The LT&RS codes have the best overall performance with a cost of the highest decoding complexity among the candidate schemes. It has a combined decoding complexity of the RS codes and the LT codes. For systems with energy-limited receivers such as the one considered in this thesis, the trade-off between the gain in channel error tolerance and the level of increase in decoding complexity should be considered. The LTAM code is an alternative choice that has many of the same properties as the LT&RS code, with a much lower decoding complexity but also less error tolerance.

Appendix A

Glossary

ACK Positive acknowledgement of a packet reception

ARQ Automatic Repeat reQuest

AWGN Additive White Gaussian Noise

BCH Bose-Chaudhuri-Hocquenghem code, a type of binary linear error correction code

BEC Binary Erasure Channel, it is used specifically as a packet level erasure channel.

BER Bit Error Rate: the probability that a received bit is in error

BPSK Binary Phase Shift Keying

BSC Binary Symmetric Channel, it is used as a bit level error channel.

Continuation Pair Two degree-2 LT encoded packets (or received packets) that contain user packets in such a way that by knowing the value of any one of the user packet, the values of the rest of the user packets can be known.

Decoding Pair Two LT encoded packets with degrees w and $w + 1$ that have w common neighbours so successful reception of both encoded packets will result in decoding of the extra user packet contained in the encoded packet with degree $w + 1$.

Degree The number of user packet(s) that are mapped to an LT encoded packet

DVB-T2 Digital Video Broadcasting – Second Generation Terrestrial

FEC Forward Error Correction

GE channel Gilbert-Elliott Channel, a two-state Markov chain that models a burst error channel

IPTV Internet Protocol Television

HDTV High Definition Television

Knee The point where the curve (specifically used for describing the success rate curve) starts to drop significantly on a graph

LAN, WLAN Local Area Network, Wireless Local Area Network

LDPC codes Low Density Parity Check codes

LT codes Includes the original LT code designed by Luby et. al. and other improvements to LT codes that inherit the rateless property, encoding method and decoding method as the original LT codes

LTAM LT codes with Added Memory. A novel LT type encoding method that was developed during the course of this thesis that reduces the decoding overhead for small message lengths over the original LT codes.

Memoryless error channel Communication channels with a constant error rate. It is used interchangeably with BSC and BEC.

NACK Negative acknowledgement of a packet reception

Neighbours The actual user packet(s) that are mapped to an LT encoded packet

PER Packet Erasure Rate, the probability of losing a packet at the receiver with a known BER and packet size

RS code Reed-Soloman code, a type of non-binary linear error correction code

Self transition probability In GE channel, it is the probability that the channel stays in the same state in the current time slot as the previous time slot.

SNR Signal to Noise Ratio

Transition Interval The range of channel error rate that the success rate transits from a stable high value to a stable low value

Transmitted Packet Encoded packets that are transmitted from the transmitter. It is mainly used for describing the repetition codes since each transmitted packet is not encoded within the packet. Therefore the term transmitted packets is a better description than encoded packets.

Received Packet Encoded (or transmitted) packets that are successfully received at the receiver

User Data Large and finite blocks/packets of information that needs to be disseminated from the transmitter to the receiver(s)

Appendix B

Additional Results for the LTAM Code

In the first three sections of this chapter, the behaviour of the LTAM method in terms of the overhead produced under various combinations of parameter settings, message lengths and channel conditions are analysed: Section B.1 provides evidence in choosing the Robust Soliton distribution for the LTAM method, Section B.2 selects the values for the parameters of the Robust Soliton distribution to be used for the problem, and Section B.3 shows the process of choosing the value p. In the final section of this chapter, Section B.4, the additional transmission cost of the original LT code over the LTAM code is examined for a range channel error rate with different code rates.

B.1 Robust vs. Ideal Soliton Distribution

Figures B.1 to B.6 compare the average overhead of the Robust and Ideal Soliton distribution produced at various p values for packet erasure rates (PER) of 0, 0.1, 0.3, 0.5, 0.7 and 0.9 for $k = 86$, 200 and 500. From Figures B.1, B.3 and B.5, it can be seen that the Robust and Ideal Soliton have similar performance for PER of 0, 0.1 and 0.3 across all values of p. However, when the erasure rate becomes larger, as can be seen from Figures B.2, B.4 and B.6, the Robust Soliton distribution starts to outperform the Ideal Soliton distribution. Because the Robust Soliton distribution has either equal or better performance than the Ideal Soliton distribution for all PER and p tested, it was decided to be used for the simulation.

B.2 Parameters of the Robust Soliton Distribution

Figure B.7 shows the overhead produced by different δ, c and p under no loss channel conditions. δ values tested are 0.1 0.3 0.5 0.7 and 0.9. c values tested are 0 0.01 0.03 0.05 0.1 0.3 0.6 and 0.9. p values tested are 0 0.05 0.1 0.15 and 0.2. The graph shows the best value of p is 0.15 (under no loss channels) and the best value of c is 0.

The best δ cannot be determined since there is no evidence indicating that a particular δ value is

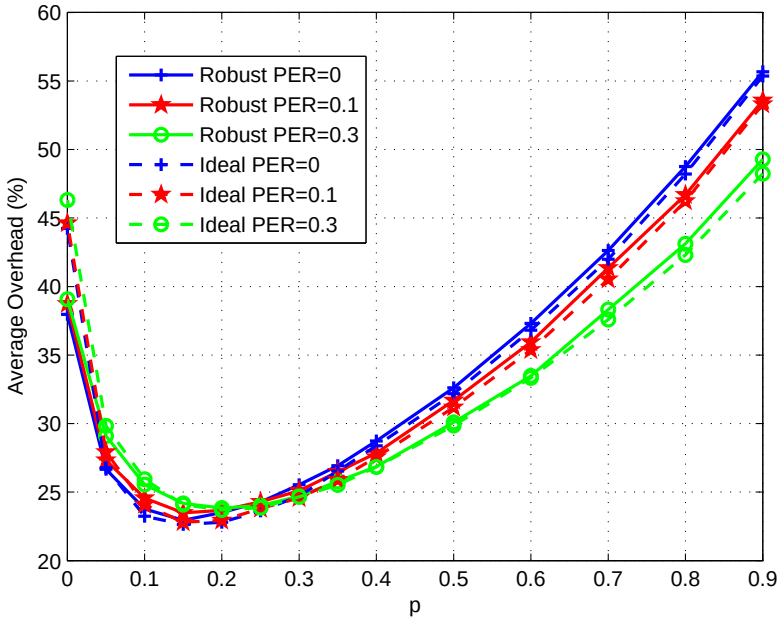

Figure B.1: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p, LTAM $k = 86$.

superior to others as shown in Figure B.9. Figure B.9 is a close up examination of the behaviour of δ with c set to 0 and the three lines represent p values of 0.1 0.15 and 0.2. Ten thousand independent experiments were executed for each point in Figure B.9 and the confidence interval is shown as the black plus signs above and below each point. The average confidence interval appearing in the figure is 0.0242%. The average overhead fluctuate over different δ values and there is no clear minimum point and the trends are different for different values of p. The choice of δ and c are determined to be 0.9 and 0.01 respectively.

B.3 Determination of the Value p

This section presents the overhead behaviour of the LTAM method under different channel erasure rates. Although Figures B.1 to B.6 already shown the behaviour of the LTAM code for different p values. This section examines the effect of different p more specifically for the Robust Soliton distribution.

Since the LTAM encoded packets are not constructed independently of each other, the overhead performance is affected by different channel loss rates and it is shown in Figure B.10: the average

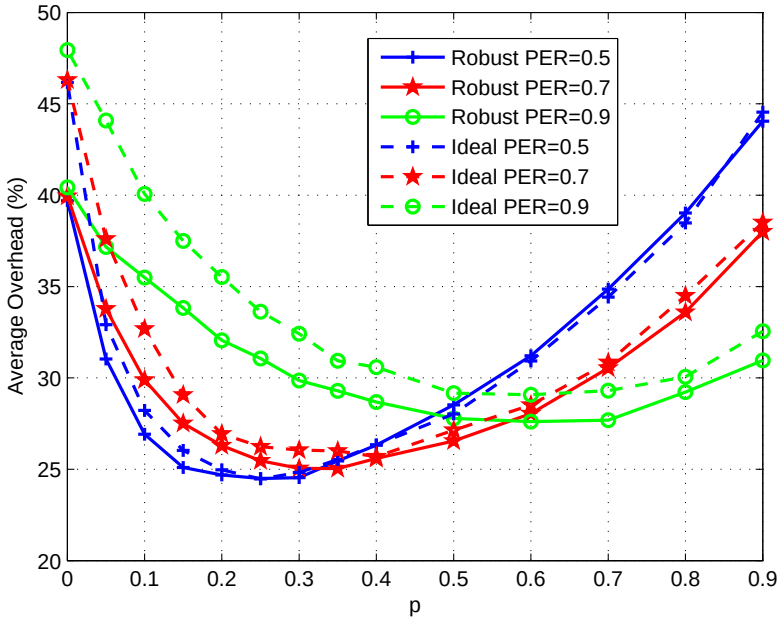

Figure B.2: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p, LTAM $k = 86$.

overhead increases as the channel loss rate increases for the LTAM code while the original LT code is not affected by different channel loss rates due to the independent construction of encoded packets. In this section the behaviour of the LTAM method under different packet erasure rates is investigated.

Figure B.11, B.12 and B.13 show the average overhead produced with different p values (across the abscissa) at different PER (shown by the different lines) for $k = 86, 200$ and 500 respectively. It can be seen from the graphs that as the PER increases, a higher p value is required to achieve the minimum overhead. However, as a general trend as the PER becomes larger, the minimum overhead that can be achieved by the LTAM method becomes larger.

Take B.11 for example. The minimum (23.5%) overhead occurs at $p = 0.15$ when PER=0.1 but with the same p value, the overhead is 33.8% for PER=0.9 while the minimum overhead that can be achieved for PER=0.9 is 27.6% at $p = 0.6$. It is desirable to determine a p value that is not in favour with a particular channel condition, but can achieve low overhead for most PERs. The following crossover points were therefore decided to be chosen: $p = 0.3$ for $k = 86$, $p = 0.2$ for $k = 200$ and $p = 0.1$ for $k = 500$.

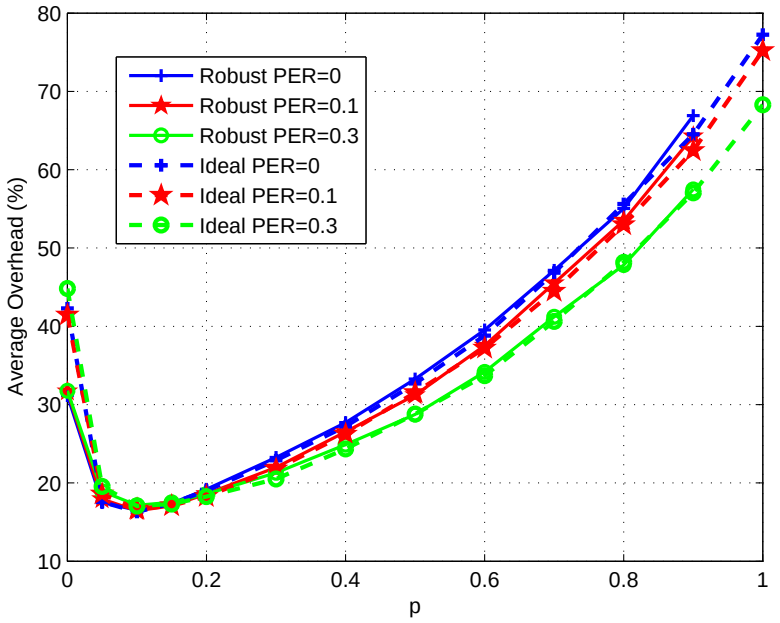

Figure B.3: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p,LTAM $k = 200$.

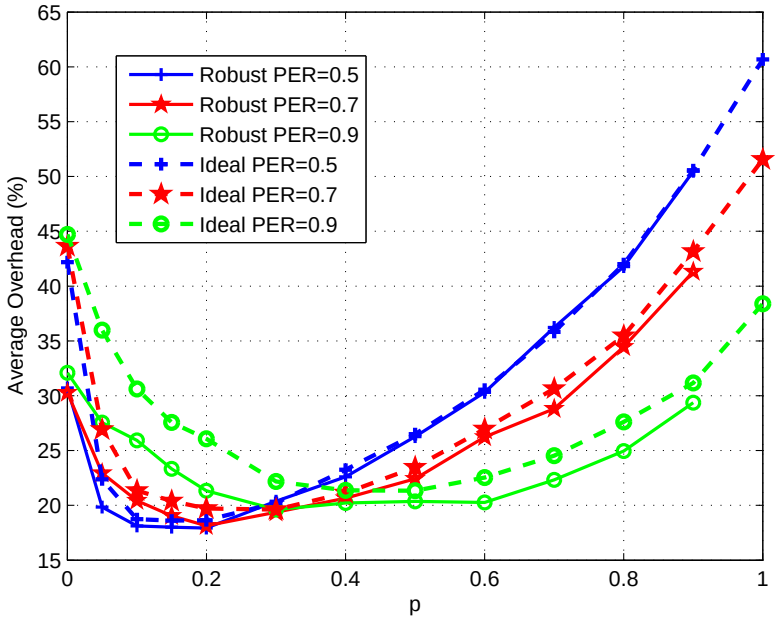

Figure B.4: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p,LTAM $k = 200$.

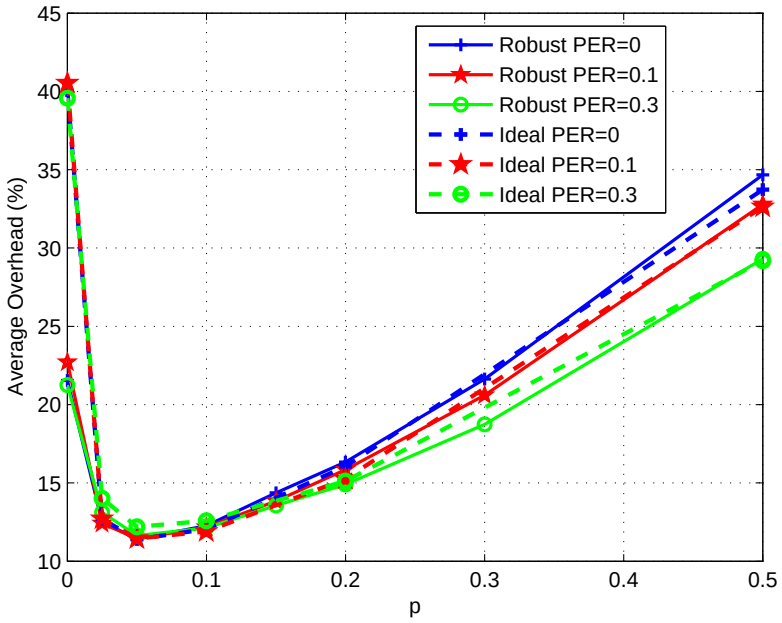

Figure B.5: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p,LTAM $k = 500$.

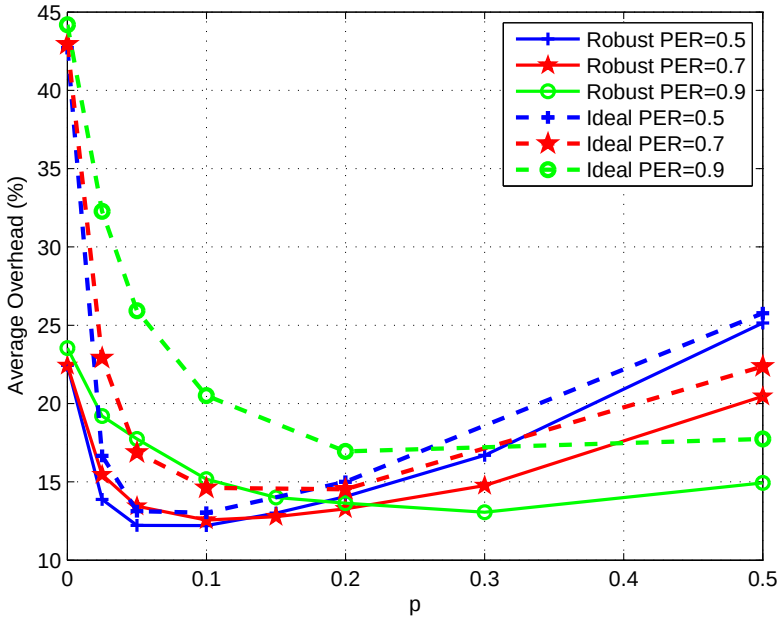

Figure B.6: Performance comparison between Robust and Ideal Soliton distribution in average overhead needed for different p,LTAM $k = 500$.

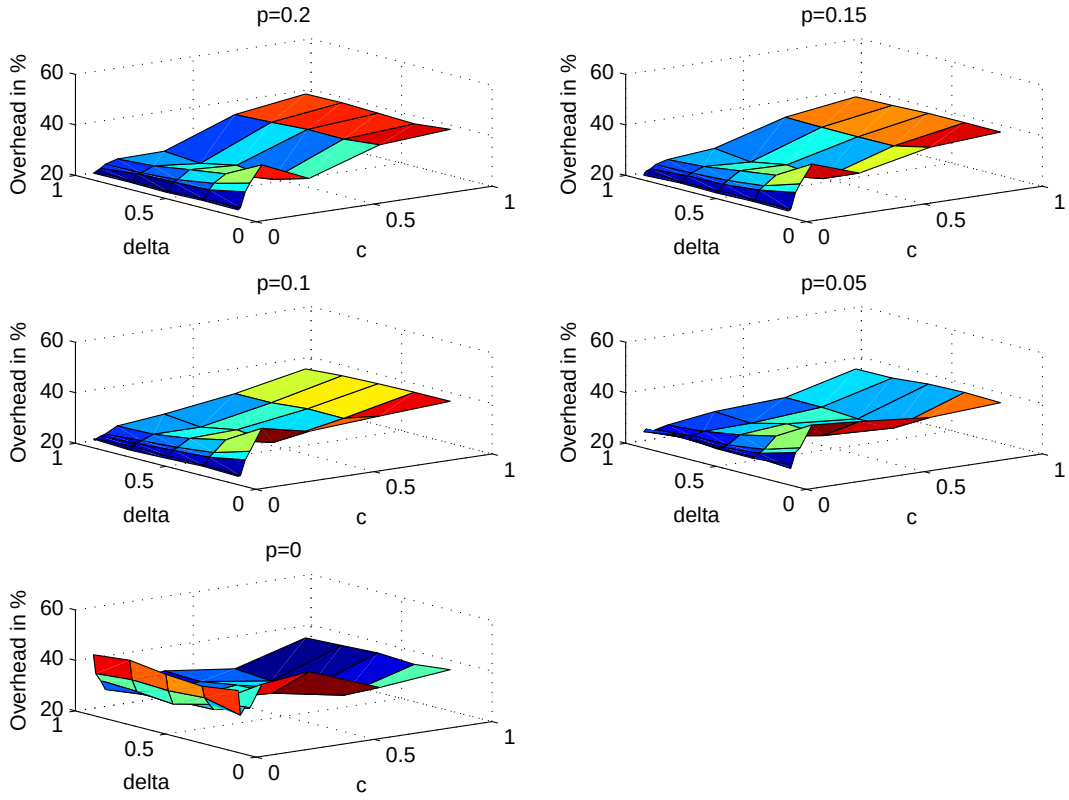

Figure B.7: Average overhead needed for different δ and c for $k = 86$.

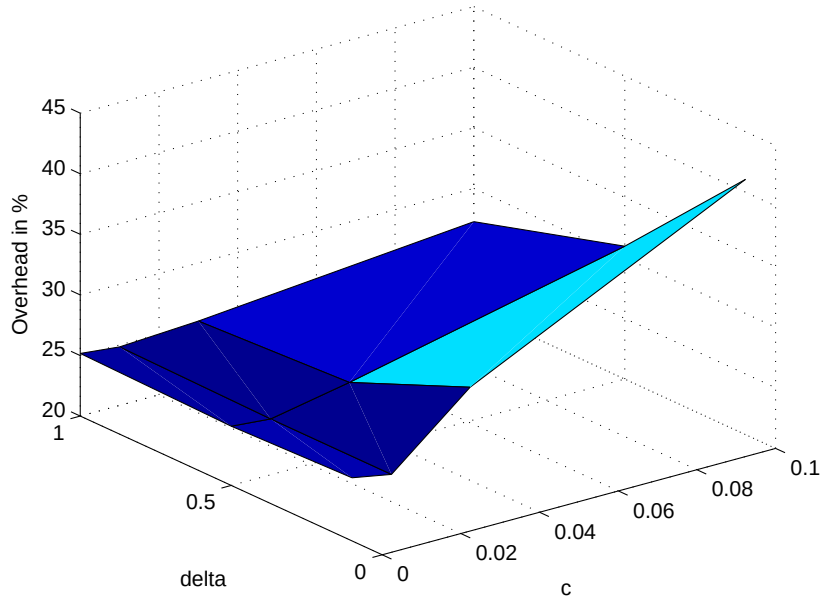

Figure B.8: Average overhead needed for different δ and c for $k = 86$ under lossy channel, BER=0.002.

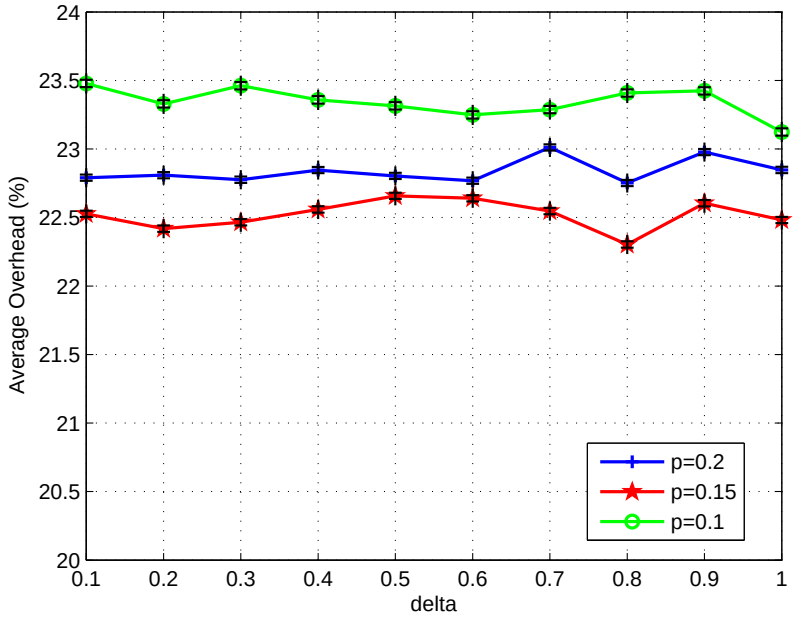

Figure B.9: Average overhead needed for different δ. LTAM $c = 0$ for $k = 86$.

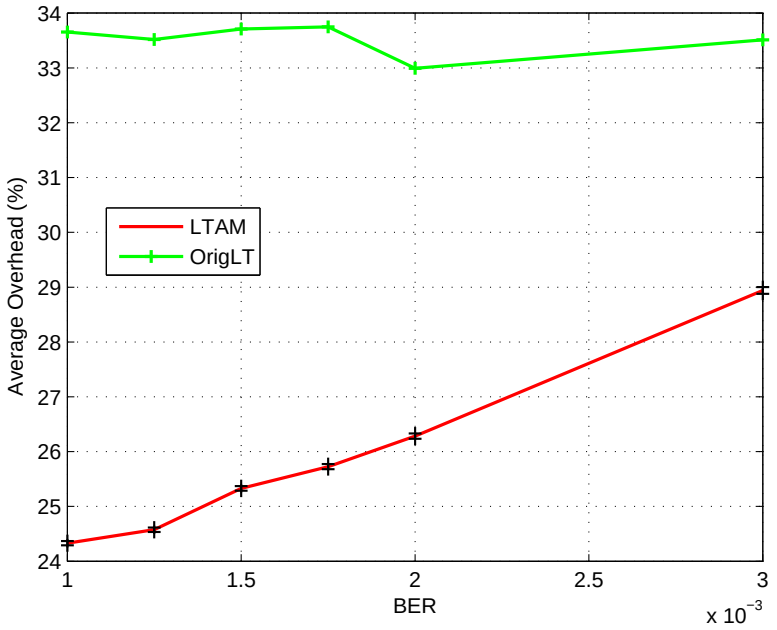

Figure B.10: Average overhead required at the receiver to decode the original LT and the LTAM encoded message with different PER for $k = 86$ and $p = 0.15$.

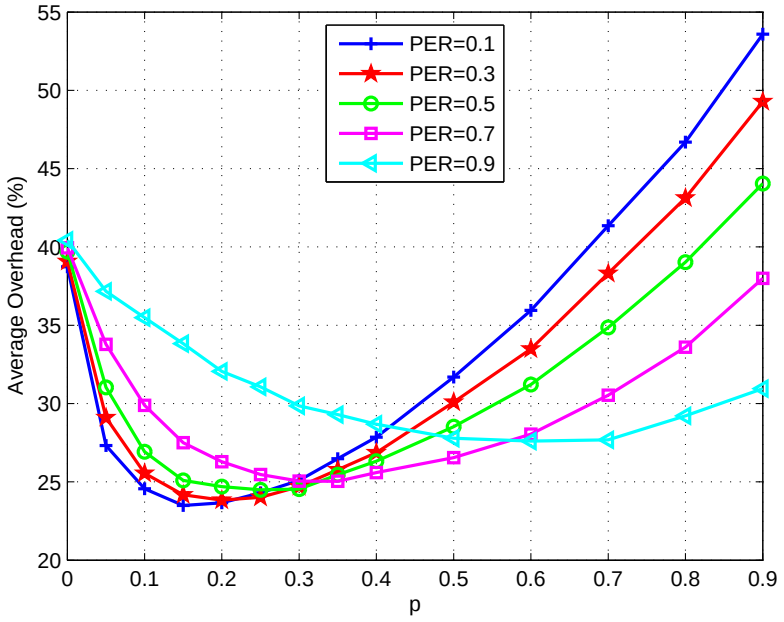

Figure B.11: Average overhead needed for different LTAM parameter p, and different PER. $k = 86$.

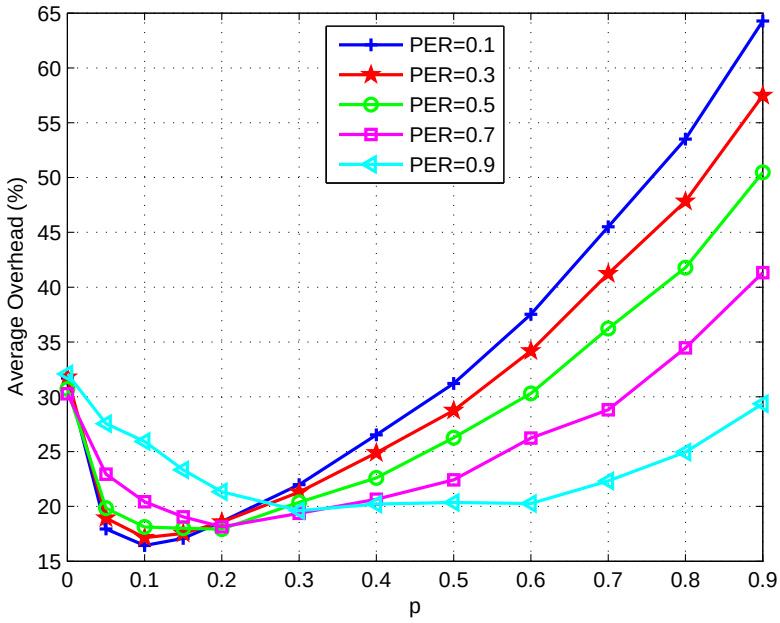

Figure B.12: Average overhead needed for different LTAM parameter p, and different PER. $k = 200$.

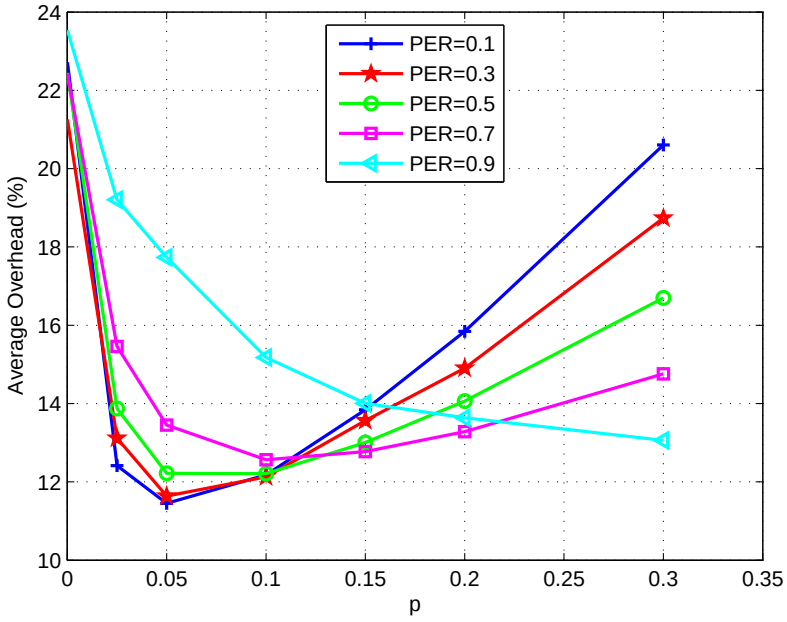

Figure B.13: Average overhead needed for different LTAM parameter p, and different PER. $k = 500$.

B.4 Comparison on the Success Rate and Transmission Cost between the LTAM and the Original LT code

Figure B.14 shows the difference between the LTAM and the original LT code in terms of transmission cost for a range of code rates. It can be seen that as the code rate becomes lower, a peak is appearing. Figure B.15 presents the corresponding received budget difference for the curves in Figure B.14. The received budget differences are constant before they decreases due to the limited transmission budget; it is expected because the received budget is a measure of overhead needed for decoding, it is a relatively constant value for both the original LT code and the LTAM code[1].

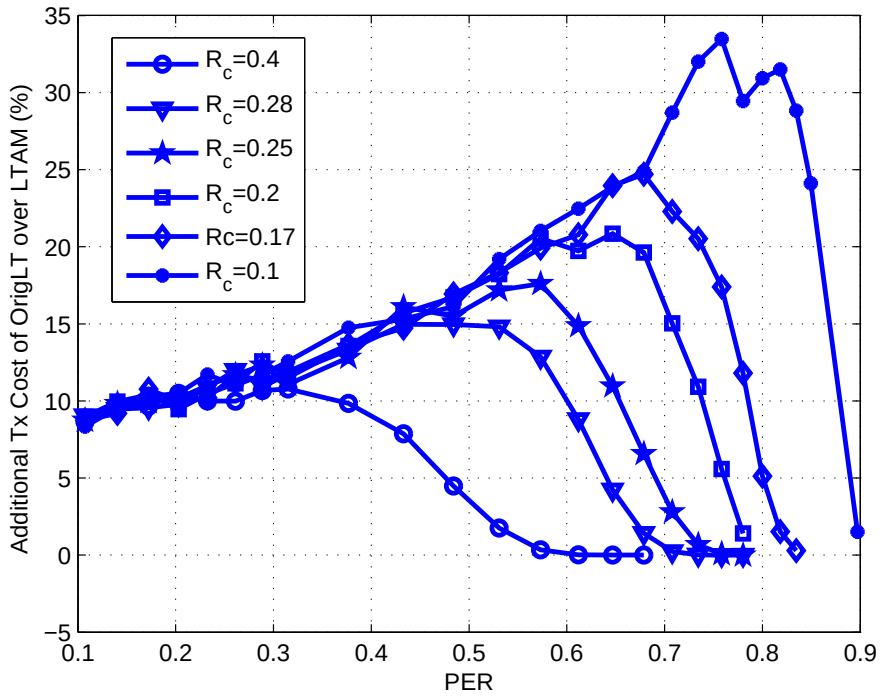

Figure B.14: Additional transmission cost of the original LT code over the LTAM code for a range of code rates (sample statistics, size: 5000).

[1] The overhead for a fixed k can vary slightly for the LTAM code over channels with very high erasure rate, see Figure B.11. However, the average overhead for the original LT code is constant for a fixed k.

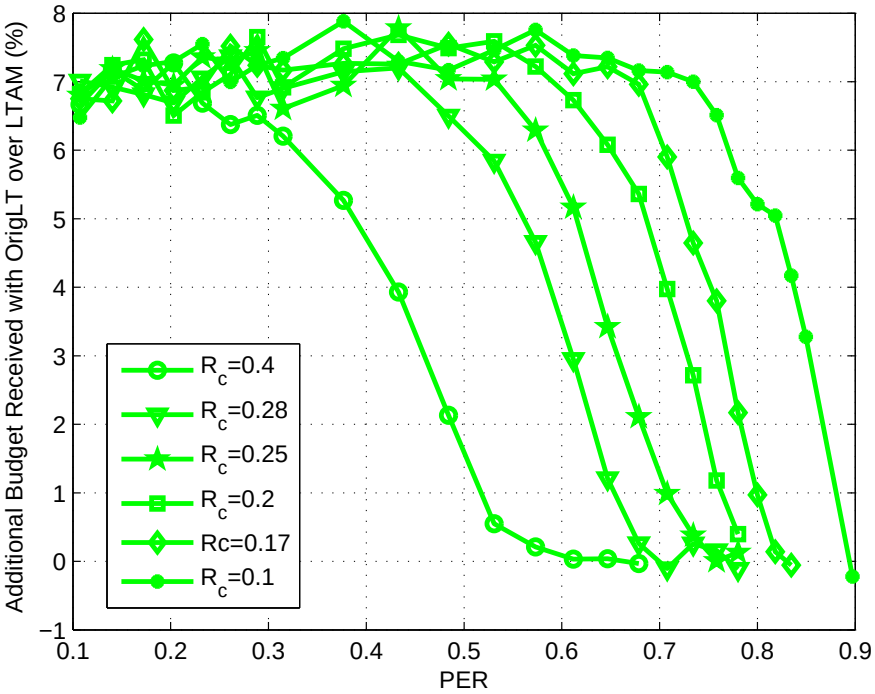

Figure B.15: Additional received budget needed by the original LT code over the LTAM code for a range of code rates (sample statistics, size: 5000).

Appendix C

List of Statistical Significance of the Figures

This appendix shows the statistical significance to most graphs in this thesis. Most of them were below 1%. Dash either means the corresponding data can not be obtained or it is not recorded.

Table C.1: Statistical Significance for Figures in Chapter 3.

Figure	**Replications per point**	**Average Confidence Interval (%)**
3.3	10,000	0.035
3.1	2000	0.088

Table C.2: Statistical Significance for Figures in Chapter 4

Figure	**Number of Accumulation of Packet Errors per Point**	**Note**
4.2	100	Monte Carlo experiment is conducted for this figure.

Table C.3: Statistical Significance for Figures in Chapter 5.

Figure	**Replications per point**	**Average Confidence Interval (%)**
5.8	5,000	0.093
5.11	100 – 10,000	0.22
5.12	100 – 10,000	0.24
5.13	100 – 10,000	0.66

Table C.4: Statistical Significance for Figures in Chapter 6.

Figure	**Scheme**	**Replications per point**	**Average Confidence Interval (%)**
6.1	RS code	–	0.009
6.30	LT code	5,000	$< 5 \times 10^{-4}$
	LTAM code	5,000	$< 5 \times 10^{-4}$
	Repetition code	2,000	–
	LT&RS code	300	0.001
6.5	RS code	–	0.005
6.32	LT code	5,000	$< 5 \times 10^{-4}$
	LTAM code	5,000	$< 5 \times 10^{-4}$
	Repetition code	2,000	–
	LT&RS code	300	< 0.001
6.13	RS code	–	0.003
6.34	LT code	5,000 – 10,000	$\sim 7 \times 10^{-4}$
	LTAM code	5,000 – 10,000	$\sim 5 \times 10^{-4}$
	Repetition code	2,000	–
	LT&RS code	1000 – 5000	$< 5 \times 10^{-4}$
6.17	RS code	–	0.002
6.36	LT code	5,000 – 10,000	$\sim 6 \times 10^{-4}$
	LTAM code	5,000 – 10,000	$\sim 4 \times 10^{-4}$
	Repetition code	2,000	–
	LT&RS code	1000 – 5000	$< 5 \times 10^{-4}$

Appendix D

Source Code

In this appendix, the important MATLAB functions for LTAM encoder and decoder are documented in Section D.1 and D.2 respectively. At the beginning of each section, a dependency graph is shown for the included functions.

D.1 LTAM Encoder

The dependency graph of the functions used for LTAM encoding is shown in Figure D.1.

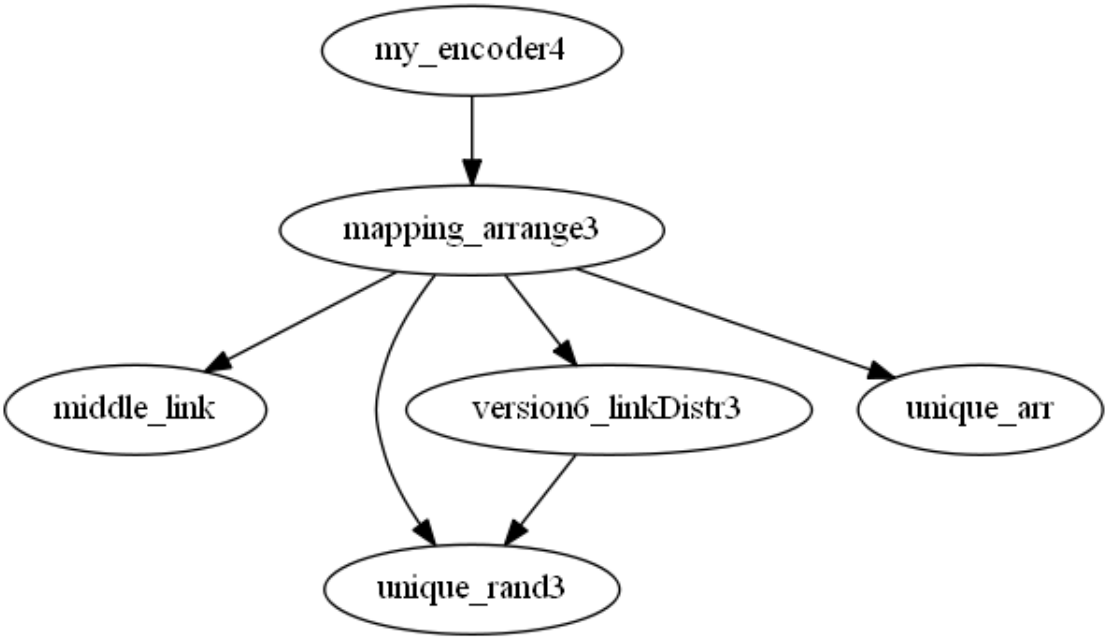

Figure D.1: An LT encoder function dependency diagram.

```
1 function [encoded_m,...
2           userPacketLinkTx,...
3           userPacketLinkRx,...
```

```
4             map_distr ,...
5             degreesTx ,...
6             degreesRx ,...
7             numPRx ,...
8             manual_map ,...
9             channelInfo ,channelMonitor] = ...
10                        my_encoder4(m,...
11                                    degreesTx ,...
12                                    channel ,...
13                                    channelInfo ,...
14                                    numPRx ,...
15                                    packetLen ,...
16                                    usrPacketNum ,...
17                                    map_distr ,...
18                                    threshold_info ,mode,channelMonitor)
19 % This function encodes user data using the LTAM encoding method.
20 % Input
21 % m                 packetized user data
22 % degreesTx         degrees for the encoded packets
23 % channel           channel type
24 % channelInfo       channel parameters
25 % numPRx            number of received packets
26 % packetLen         packet length
27 % usrPacketNum      number of user packets
28 % map_distr         an example at the end of function mapping_arrange3
29 % threshold_info
30 % mode              encoding mode 2 for an ancestor of LTAM, 3 for LTAM
31 % channelMonitor    monitors the channel state
32 %
33 %
34 % Output
35 % encoded_m         encoded message
36 % userPacketLinTx   mapping of the transmitted encoded packets
37 % userPacketLinRx   mapping of the received encoded packets
38 % map_distr
39 % degreesTx         degrees of the transmitted encoded packets
40 % degreesRx         degrees of the received encoded packets
41 % numPRx            number of received packets
42 % manual_map        a collection of which encoded packets has degree 1,
43 %                   degree 2...
44 % channelInfo       channel parameters, this parameter is passed out
45 %                   because the state that the channel is in should be
46 %                   maintained
47 % channelMonitor    monitors the channel state
48 %
49 % Comment updated on        29/May/2012      by Sasha Wang
50 % Comment updated on        28/Feb/2013      by Sasha Wang
51 header_file;
52 %% Encoding Process
53
54 % Pre-allocations
```

```
55 userPacketLinkTx=cell(1,usrPacketNum);
56 manual_map = cell(1,usrPacketNum);
57
58
59 userPacketLinkRx=cell(1,0);
60 encoded_m = ones(0,packetLen)*(-1);
61 degreesRx = ones(1,0)*(-1);
62
63 if channel == BEC
64     PsucBEC = channelInfo;
65 elseif channel == GE
66     currState = channelInfo(5);
67 end
68
69
70 for i = 1:usrPacketNum
71     % This produces the candidate positions for the encodedPacket to map to
72     % the message m.
73
74
75     if mode == 2 % This method is a pre-LTAM method
76         [userPacketLinkTx{i},...
77             map_distr,...
78             degreesTx(1:i),...
79             manual_map] = mapping_arrange2...
80                                         (usrPacketNum,...
81                                         degreesTx(1:i),...
82                                         map_distr,...
83                                         userPacketLinkTx,...
84                                         manual_map,...
85                                         threshold_info);
86     elseif mode == 3 % This is the LTAM method
87         [userPacketLinkTx{i},...
88          map_distr,...
89          degreesTx(1:i),...
90          manual_map] = mapping_arrange3...
91                                       (usrPacketNum,...
92                                        degreesTx(1:i),...
93                                        map_distr,...
94                                        userPacketLinkTx,...
95                                        manual_map,...
96                                        threshold_info);
97     end
98
99
100    if channel == BEC
101        Psuc = PsucBEC;
102    elseif channel == GE
103        if currState == GOOD
104            Psuc = channelInfo(2); % channelInfo(2)=gPsuc
105            if rand > channelInfo(1); % channelInfo(1)=g2g
```

```
106                 currState = BAD;
107             end
108         elseif currState == BAD
109             Psuc = channelInfo(4); % channelInfo(4)=bPsuc
110             if rand > channelInfo(3); % channelInfo(3)=b2b
111                 currState = GOOD;
112             end
113         end
114     end
115
116     % The process below determines which packets received at the receiver.
117     if rand < Psuc
118         channelMonitor(i) = 1;
119         numPRx = numPRx + 1;
120         userPacketLinkRx{numPRx} = userPacketLinkTx{i};
121         encoded_m(numPRx,:) = m(userPacketLinkTx{i}(1),:);
122         degreesRx(numPRx) = degreesTx(i);
123         for j = 2:degreesRx(numPRx)
124             encoded_m(numPRx,:) = bitxor(encoded_m(numPRx,:),...
125                                 m(userPacketLinkRx{numPRx}(j),:));
126         end
127     else
128         channelMonitor(i) = 0;
129     end
130 end
131
132 if channel == GE % Preserve the current state of the GE channel
133     channelInfo(5) = currState;
134 end
```

```
1  function [currMapping map_distr degrees manual_map] = mapping_arrange3...
2              (usrPacketNum,degrees,map_distr,userPacketLink,manual_map,...
3              threshold_info)
4  % This function applies the LTAM mapping method
5  %
6  % Input
7  % degrees:       array of degrees, the last element is on the encoded
8  %                 packet that are about to generate now.
9  % map_distr:     distribution of the mapping, an example at the end
10 %                of this file.
11 % manual_map:    information for manual_map, an example at the end of this
12 %                file
13 % Output
14 % currMapping:   the user packet that the current encoded packet
15 %                is linked to
16 % map_distr:     updated map_distr
17 % degrees:       updated degrees
18 % manual_map:    updated manual_map
19 %
20 % File created on 19/Mar/2012        by Sasha Wang
21 % Comment updated on 25/May/2012           "
22 % File updated on 10/Sep/2012             "
23
24 header_file;
25
26 moreThan3thresh = threshold_info(1);
27 lessThan3randThresh = threshold_info(2);
28 lessThan3lengthThresh = threshold_info(3);
29 missEleThresh = threshold_info(4);
30
31 currDegree = degrees(end); %last element
32 currIndex = length(degrees);
33 same = 1;
34
35 i = 1;
36 while same == 1
37     if currDegree < 3
38         % when degree is 2 AND
39         % when "lessThan3randThresh" (0.5) of chance AND
40         % when the number of degree 2 encoded packet exceed
41         % "lessThan3lengthThresh"(0.5)
42         if currDegree == 2 & rand(1)>lessThan3randThresh & ...
43           length(manual_map{2})>lessThan3lengthThresh*usrPacketNum
44             currMapping = middle_link(usrPacketNum,currDegree,...
45                                 userPacketLink,manual_map,map_distr);
46         % when degree is 1 AND
47         % when "lessThan3randThresh" (0.5) of chance AND
48         % when the number of degree 2 encoded packet exceed
49         % "lessThan3lengthThresh"(0.5)
50         % the last two conditions are the same as above
51         elseif currDegree == 1 & rand(1)>lessThan3randThresh & ...
```

```
52              length(manual_map{2})>lessThan3lengthThresh*usrPacketNum
53                  index = randi(length(manual_map{2}),1);
54                  currMapping = userPacketLink{index}(1);
55              else
56                  [currMapping map_distr] = unique_rand3(usrPacketNum,...
57                          currDegree,currIndex,map_distr,missEleThresh);
58              end
59
60          else
61              my_rand = rand(1);
62              if my_rand > moreThan3thresh
63                  [currMapping map_distr] = version6_linkDistr3...
64                      (degrees,currDegree,userPacketLink,map_distr,...
65                       usrPacketNum,currIndex,manual_map,missEleThresh);
66              else
67                  [currMapping map_distr] = unique_rand3(usrPacketNum,...
68                          currDegree,currIndex,map_distr,missEleThresh);
69              end
70          end
71          same = unique_arr(userPacketLink,usrPacketNum,...
72                                  currMapping,length(degrees));
73          if currMapping == usrPacketNum | i > 3
74              break;
75          end
76          i = i+1;
77      end
78
79      % update map_distr
80      for i=1:currDegree
81          map_distr(currMapping(i)) = map_distr(currMapping(i))+1;
82      end
83
84
85
86      manual_map{currDegree}(end+1) = length(degrees);
87
88
89      %% Debug message
90      if length(currMapping) ~= currDegree
91          rand_num_len = length(currMapping)
92          currDegree
93          error('rand_num len doesn''t match degree!!!')
94      end
95
96      degrees(length(degrees)) = currDegree;
97
98
99      % manual_map example------
100     % userPacketNum = 5
101     % degree
102     %   1     {[]
```

```
103 %    2       [1 3 5 6 7 8 10]  <- This means encoded packets 1,3,5...10
104 %                                                    have degree 2
105 %    3       []
106 %    ...
107 %    5      []}
108 %-------------------------
109
110 % map_distr example--------
111 % map_distr =[7 3 7 4 3 7 6 4 9 5];
112 %
113 % This means 7 encoded packet included user packet 1
114 % 3 encoded packets included user packet 2 ...
```

```
1  function currMapping = middle_link(usrPacketNum,currDegree,...
2                       userPacketLink,manual_map,map_distr)
3  % The function middle_link constructs a degree two mapping with
4  % one maps to one of user packet that is included in another
5  % degree two encoded packet and the other one maps to a userpacket
6  % that has not been connect if there is one such a user packet.
7  % Otherwise,the second mapping links to a user packet that has
8  % the least link.
9  %
10 % Input
11 % userPacketLink: all the existing user packet links so far
12 % manual_map:      see example in mapping_arrange3.m
13 % map_distr:       see example in mapping_arrange3.m
14 %
15 % Output
16 % currMapping: the user packet that the current encoded packet
17 %               is linked to
18 %
19 %
20 % Comment added on 2/Nov/2012        by Sasha Wang
21 %==========================================================================
22
23 assert(currDegree == 2,'CurrentDegree is not 2');
24 n = length(manual_map{2}); %number of encoded packets with degree
25                            % 2 in manual_map
26 index = randi(n,1); % randomly select one index for userPacketLink
27
28 whichEncodedP = manual_map{2}(index); % which encoded packet is selected.
29
30 m = length(userPacketLink{whichEncodedP}); % must be 2 right?
31
32 index2 = randi(m,1); % randomly select one index for a mapping
33                       % in userPacketLink{index}
34
35 ele1 = userPacketLink{whichEncodedP}(index2);
36
37 missingEle = find(map_distr==0); % find if there is any user
38     % packet has not been included in all the encoded packets
39 if ~isempty(missingEle)
40     ele2 = missingEle(1);
41 else
42     minUsrPakInd = find(map_distr==min(map_distr)); % find the
43                     % user packet that has been least mapped to
44     ele2 = minUsrPakInd(1);
45     while ele2 == ele1
46         ele2 = randi(usrPacketNum,1); % ele2 must not be equal to ele1
47     end
48 end
49
50 currMapping = [ele1 ele2];
51 currMapping = sort(currMapping);
```

```
function [currMapping map_distr] = unique_rand3...
    (usrPacketLen,currDegree,currIndex,map_distr,missEleThresh)
% This function unique_rand3 ensures the case 4 of the LTAM code
% construction.

header_file;
currMapping = -1*ones(1,currDegree);

start = 1;

% if the current encoded packet exceeded k (number of user packet)
% find if there is any user packet has has not been linked
% if so, make the fist mapping to that unlinked user packet
if currIndex > usrPacketLen
    missingEle = find(map_distr==0);
    if ~isempty(missingEle)
        currMapping(1) = missingEle(randi(length(missingEle),1));
        start = 2;
    end
% if the current encoded packet hasn't exceeded k, but it has exceeded
% "missEleThresh"(0.97), also find if any user packet is unlinked, if so,
% link it.
else
    if currIndex>missEleThresh*usrPacketLen && currIndex < usrPacketLen
        missingEle = find(map_distr==0);
        if ~isempty(missingEle)
            currMapping(1) = missingEle(randi(length(missingEle),1));
            start = 2;
        end
    end
end

% for the rest of the links, do random mapping
for i=start:currDegree
    n = randi(usrPacketLen,1);
    max_map = -1;
    while ismember(n,currMapping) | n==max_map
        n = randi(usrPacketLen,1);
    end
    currMapping(i) = n;
    for j=1:i-1
        if n == currMapping(j)
            error('n is the same as one prev mapping!!!')
        end
    end
end

currMapping = sort(currMapping);
```

```
function [currMapping map_distr] = version6_linkDistr3...
    (degrees,currDegree,userPacketLink,map_distr,usrPacketNum,...
    currIndex,manual_map,missEleThresh)
% This function find a previous encoded packet with a degree
% that is 1 less than the currDegree


% place the index of the degree into the corresponding slot in manual_map
% manual_map{currDegree} = length(degrees);
% preallocate
currMapping = zeros(1,currDegree);

if ~isempty(manual_map{currDegree-1})
    corr_rand = randi(length(manual_map{currDegree-1}),1); % which
                                %encoded packet is to correlate with
    currMapping(1:currDegree-1) = ...
                userPacketLink{manual_map{currDegree-1}(corr_rand)};
    n = randi(usrPacketNum,1);
    while ismember(n,currMapping)
        n = randi(usrPacketNum,1);
    end
    currMapping(currDegree) = n;
    currMapping = sort(currMapping);

else
    [currMapping map_distr] = unique_rand3(usrPacketNum,currDegree,...
                                currIndex,map_distr,missEleThresh);
end
```

```
function same = unique_arr(userPacketLink,packNum,newArr,currMapStage)
% This function checks if the newly added cell to userPacketLink
% is the same as any previous cell in userPacketLink.
same = 0;
for j = 1:currMapStage-1
    % If the length of the new array is the same as one of
    % the existing one
     if length(userPacketLink{j})==length(newArr)
         if length(userPacketLink{j}) == packNum
             userPacketLink{j}
             disp('same number in degree with max length')
             same = 0; % this is not correct have a look at it later
         else
             if userPacketLink{j}==newArr
                same = 1;
                break;
             end
         end
     end
end
```

D.2 LT Decoder

The dependency graph of the functions used for LTAM decoding is shown in Figure D.2.

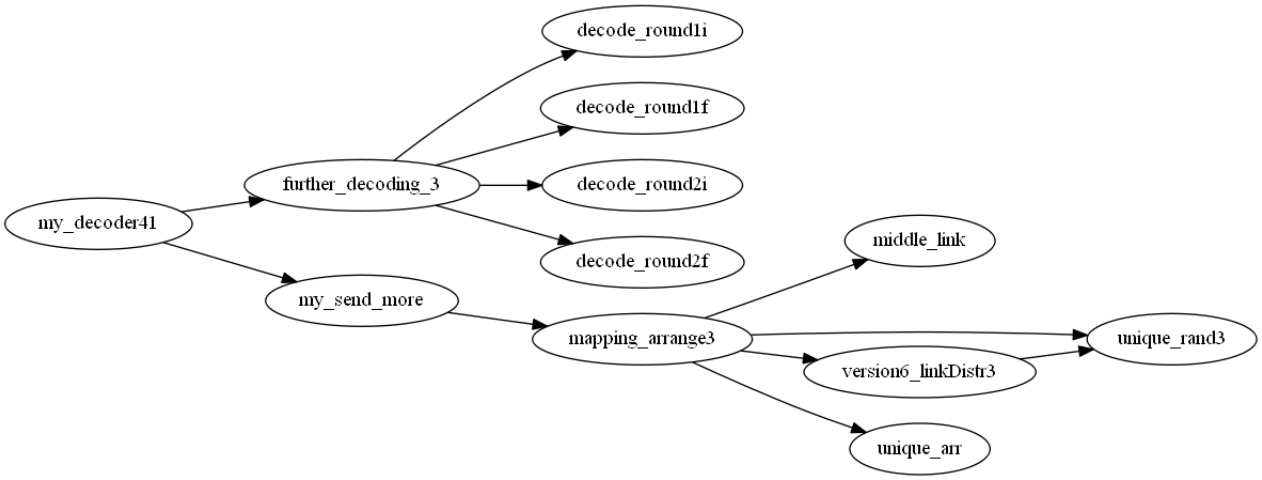

Figure D.2: An LT decoder function dependency diagram.

```
function [decodedData,...
          numPTx,...
          numPRx,...
          userPacketLinkTx,...
          userPacketLinkRx,...
          degreesTx, ...
          degreesRx, ...
          manual_map map_distr,channelMonitor] = ...
                                        my_decoder41...
                                         (userPacketLinkTx,...
                                          userPacketLinkRx,...
                                          degreesTx,...
                                          degreesRx,...
                                          encoded_m,...
                                          usrPacketNum,...
                                          addnInfo,...
                                          numPTx,...
                                          numPRx,...
                                          channel,...
                                          channelInfo,...
                                          runningMode,...
                                          map_distr,...
                                          manual_map,...
                                          threshold_info,...
                                          Rc,mode,channelMonitor)
% This function decodes the LT code.
%
```

```
28 % Inputs:-----------------------------------------------------------------
29 % userPacketLink:    cell type also length of TESTING_LENGTH, each
30 %                    element contains an array of the indeces that the
31 %                    particular encoded packet
32 %                    links to.
33 % encoded_m:         the actual encoded message.
34 % usrPacketNum:      number of user packet.
35 % packetLen:         number of bit per packet.
36 % addnInfo:          additional info needed for acquiring more packets
37 %                    from the transmitter e.g. the distribution,
38 %                    user packets.
39 % numPtx:            number of packets transmitted - this is used for
40 %                    packet loss rate.
41 % Psuc:              success reception probability.
42 % degreeMappingVec: vector, indicates how many user packets each encoded
43 %                    packet link to.
44 % runningMode:       NORM: for normal, DBG for debugging.
45 %
46 % Output:----------------------------------------------------------------
47 % decodedData:       the decoded message
48 % numPTx:            number of transmitted packets
49 % numPRx:            number of received packets
50 % userPacketLinkTx: cell structure, the neighbours of each transmitted
51 %                    encoded packet.
52 % userPacketLinkRx: cell structure, the neighbours of each received packet.
53 % degreesTx:         degrees of the transmitted encoded packets.
54 % degreesRx:         degrees of the received packets.
55 %
56 % File Created on       14/Mar/2012   by Sasha Wang
57 % Updated on            31/May/2012        "
58 % Comments updated on   23/Jun/2012        "
59 %-------------------------------------------------------------------------
60 header_file;
61 % dummyMat is the decoding matrix-----------------------------------
62 % -2 indicate there is no data link at that element position     |
63 % -1 indicate there is a link and data at that element position |
64 % 0 or 1 indicate the actual data at that position----------------
65 dummyMatRx = -2*ones(length(userPacketLinkRx),usrPacketNum);
66 for i = 1:length(userPacketLinkRx)
67     for j = 1:length(userPacketLinkRx{i})
68         dummyMatRx(i,userPacketLinkRx{i}(j)) = -1;
69     end
70 end
71
72 % initialise decodedData to be all -1s.
73 decodedData = -1*ones(1,size(dummyMatRx,2));
74 changedRows = zeros(1,10);
75 arr = cell(1,usrPacketNum);
76
77 % First Decoding
78 [dummyMatRx decodedData changed needMore changedRows arr] = ...
```

```
79      further_decoding_3(dummyMatRx,encoded_m,decodedData,...
80      degreesRx,INITIAL_CALL,changedRows,arr);
81  % DBG_decode_iter = 1;
82  while changed == 1
83      [dummyMatRx decodedData changed needMore changedRows arr] = ...
84          further_decoding_3(dummyMatRx,encoded_m,decodedData,...
85          degreesRx,FOLLOWING_CALL,changedRows,arr);
86  end
87
88  if runningMode==DBG
89  %      numPtx=-1;
90  else
91      % More transmission(s) needed
92      while needMore == 1 && numPTx < usrPacketNum/Rc
93          [dummyMatRx,...
94           encoded_m,...
95           userPacketLinkTx,...
96           userPacketLinkRx,...
97           degreesTx,...
98           degreesRx,...
99           numPTx,...
100          numPRx,...
101          map_distr,...
102          manual_map,channelInfo,channelMonitor]=my_send_more...
103                                  (dummyMatRx,...
104                                   encoded_m,...
105                                   addnInfo,...
106                                   userPacketLinkTx,...
107                                   userPacketLinkRx,...
108                                   degreesTx,...
109                                   degreesRx,...
110                                   numPTx,...
111                                   numPRx,...
112                                   channel,...
113                                   channelInfo,...
114                                   map_distr,...
115                                   manual_map,...
116                                   threshold_info,mode,channelMonitor);
117          if length(encoded_m) > 0
118              [dummyMatRx decodedData changed needMore changedRows arr]...
119                 = further_decoding_3(dummyMatRx,encoded_m,decodedData,...
120                 degreesRx,FOLLOWING_CALL,changedRows,arr);
121              while changed == 1
122                  [dummyMatRx decodedData changed needMore changedRows arr]...
123                  = further_decoding_3(dummyMatRx,encoded_m,decodedData,...
124                  degreesRx,FOLLOWING_CALL,changedRows,arr);
125              end
126          end
127      end
128  end
129  end
```

```
function [dummyMatRx,...
          encoded_m,...
          userPacketLinkTx,...
          userPacketLinkRx,...
          degreesTx,...
          degreesRx,...
          numPTx,...
          numPRx,...
          map_distr,...
          manual_map,channelInfo,channelMonitor]=my_send_more...
                                        (dummyMatRx,...
                                         encoded_m,...
                                         addnInfo,...
                                         userPacketLinkTx,...
                                         userPacketLinkRx,...
                                         degreesTx,...
                                         degreesRx,...
                                         numPTx,...
                                         numPRx,...
                                         channel,...
                                         channelInfo,...
                                         map_distr,...
                                         manual_map,...
                                         threshold_info,mode,channelMonitor)
% This function sends one more encoded packet "on the fly"
% to the decoder to help the decoding process.
%
% Input needed at the receiver
% dummyMat:      current (partially completed) dummy matrix
% encoded_m:     current encoded message array
%
% Input needed for forming more encoded packet at the encoder
% addnInfo
% m:                  the original message
% cumulative_distr:   distribution
% packetNum:          number of userPacket

% disp('In send_more')
header_file;

m = addnInfo.m;
cumulative_distr = addnInfo.cumulative_degree_distr;
usrPacketNum = addnInfo.usrPacketNum;



dLen=length(degreesTx);

% txlen = length(userPacketLinkTx);

L=rand(1); % Generate a new random number between 0-1
```

```
52 degreesTx(dLen+1) = degreeGen(L,cumulative_distr); % map it to
53                                     % a soliton random variable
54
55 if mode == 2
56     % At this point length of degree is 1 element more than
57     % userPacketLinkTx
58     [userPacketLinkTx{dLen+1} map_distr degreesTx manual_map] = ...
59           mapping_arrange2(usrPacketNum,degreesTx,map_distr,...
60           userPacketLinkTx,manual_map,threshold_info);
61
62 elseif mode == 3
63     % At this point length of degree is 1 element more than
64     % userPacketLinkTx
65     [userPacketLinkTx{dLen+1} map_distr degreesTx manual_map] = ...
66         mapping_arrange3(usrPacketNum,degreesTx,map_distr,...
67         userPacketLinkTx,manual_map,threshold_info);
68 end
69
70 numPTx = numPTx + 1;
71
72 assert(numPTx == dLen+1,'numPtx should now the same as dLen+1')
73
74
75 if channel == BEC
76     Psuc = channelInfo;
77 elseif channel == GE
78     currState = channelInfo(5);
79     if currState == GOOD
80         Psuc = channelInfo(2); % channelInfo(2)=gPsuc
81         if rand > channelInfo(1); % channelInfo(1)=g2g
82             currState = BAD;
83         end
84     elseif currState == BAD
85         Psuc = channelInfo(4); % channelInfo(4)=bPsuc
86         if rand > channelInfo(3); % channelInfo(3)=b2b
87             currState = GOOD;
88         end
89     end
90     channelInfo(5) = currState;
91 end
92
93
94 if rand < Psuc
95     channelMonitor(end+1) = 1;
96     numPRx = numPRx + 1;
97     degreesRx(numPRx) = degreesTx(numPTx);
98     userPacketLinkRx{numPRx} = userPacketLinkTx{end};
99     newRow = -2*ones(1,usrPacketNum);
100
101     newPacket=0; %anything xor with 0 is itself,so initiate it to 0;
102     for i=1:degreesRx(numPRx)
```

```
103             newRow(userPacketLinkRx{numPRx}(i))=-1;
104             newPacket=xor(newPacket,m(userPacketLinkRx{numPRx}(i)));
105         end
106
107         dummyMatRx(numPRx,:) = newRow;
108         encoded_m(numPRx) = newPacket;
109     else
110         channelMonitor(end+1) = 0;
111     end
112
113 end
```

```
1  function [dummyMat decodedData changed needMore changedRows arr]...
2      = further_decoding_3(dummyMat,encoded_m,decodedData,...
3                  degreeMappingVec,callType,changedRows,arr)
4  % This function further decodes the dummyMatrix
5  %
6  % Decoding Round 2 added on          23/June/2012     by Sasha Wang
7  header_file
8
9  changed = 0;
10
11
12 % disp('-------------------------------------------------')
13 % dummyMat_before_decoding = dummyMat
14 % decodedData_before_decoding = decodedData'
15 % fprintf('entering further_decoding_2 ----------\n')
16
17 %% Decoding. Going through the rows of dummyMat
18 % fprintf('decoding round 1\n');
19 if callType == INITIAL_CALL
20     [changed,dummyMat,encoded_m,decodedData] = decode_round1i...
21         (changed,dummyMat,encoded_m,decodedData,degreeMappingVec);
22     index = 1;
23 elseif callType == FOLLOWING_CALL
24     index = 1;
25     [changed,dummyMat,encoded_m,decodedData changedRows index] = ...
26     decode_round1f(changed,dummyMat,encoded_m,decodedData,...
27     degreeMappingVec,changedRows,index);
28
29 end
30
31 %% Reset changedRows
32 %
33 % changedRows = zeros(1,10);
34
35 %% Decoding Round 2 -- finds two non-decoded row with one different position
36 % and do corresponding decoding
37 % fprintf('decoding round 2\n');
38 switch callType
39     case INITIAL_CALL
40         for i = 1:length(encoded_m)
41 %           degreeMappingVec(i)
42             arr{degreeMappingVec(i)}(end+1)=i;
43         end
44         [changed dummyMat decodedData changedRows index arr]=...
45             decode_round2i(encoded_m,decodedData,degreeMappingVec,...
46             dummyMat,changedRows,index,arr,changed);
47
48     case FOLLOWING_CALL
49         if degreeMappingVec(end)>1 % only more than degree
50                                    % 1 goes in to this function
51             [changed dummyMat decodedData changedRows index arr]=...
```

```
52                  decode_round2f(encoded_m,decodedData,...
53                  degreeMappingVec,dummyMat,changedRows,index,arr,...
54                  changed);
55          end
56  end
57
58
59  % dummyMat_before_cleaning = dummyMat
60  % decodedData_before_cleaning = decodedData'
61
62  %% Clean up, make sure known elements are updated across all row
63  % Going through the columns of dummyMat if there is one known, the rest of
64  % unknown in that column becomes known.
65  % fprintf('cleaning up\n')
66  for j=1:size(dummyMat,2)
67      % position(s) of the unknown(s) in that column
68      unknownPos=find(dummyMat(:,j)==-1);
69      % position(s) of the known(s) in that column
70      knownPos= find(dummyMat(:,j)==1 | dummyMat(:,j)==0);
71      if ~isempty(unknownPos) && ~isempty(knownPos)
72          for m=1:length(unknownPos)
73              dummyMat(unknownPos(m),j) = dummyMat(knownPos(1),j);
74              changedRows(index) = unknownPos(m);
75              index = index + 1;
76          end
77      end
78  end
79  % changed
80  % dummyMat_after_cleaning = dummyMat
81  % decodedData_after_cleaning = decodedData'
82  % disp('-------------------------------------------------')
83  % pause
84
85  %% Determine if more data is needed
86  % changed == 0 &&
87  if ~isempty(find(decodedData==-1))
88      needMore = 1;
89  else
90      needMore = 0;
91  end
92
93  % decodedData
94  % needMore
95  % changed
96  % dummyMat
97  % fprintf('----------------exiting further_decoding_2\n')
98  % pause;
```

```
function [changed,dummyMat,encoded_m,decodedData] = decode_round1i...
              (changed,dummyMat,encoded_m,decodedData,degreeMappingVec)
% Decode round 1 for the first k encoded packets i.e. INITIAL_CALL
%
% Comment added      16/March/2013    by Sasha Wang

for i=1:length(encoded_m)
    %% Decoding Round 1 ---
    % (Simple Decoding) dealing with rows with one unknown
    decodeM = find(dummyMat(i,:) == 1 | dummyMat(i,:)==0);
    if isempty(decodeM)
        % if the ith row has no packet being decoded
        if degreeMappingVec(i)==1
            % if there is only one unknown link to a packet
            decodePsb = find(dummyMat(i,:) == -1);
            % replace the degree one position with the enocode packet
            dummyMat(i,decodePsb)=encoded_m(i);
            % This is a check for either the about to be changed
            % decodedData position contains either -1(haven't
            % decoded) or the same value as it been previously
            % decoded
            if decodedData(decodePsb)==-1 || ...
                    decodedData(decodePsb)==encoded_m(i)
                decodedData(decodePsb)=encoded_m(i);
            % decodePack records the position of the newly
            % knownPacket(first element) and the value (second element).
            else
                dummyMat
                decodedData(decodePsb)
                encoded_m(i)
                error('decoding data does not agree!!!')
            end
            changed = 1;
        end
    else % else: if the ith row already been partially decoded
        decodePsb = find(dummyMat(i,:) == -1);
        if length(decodePsb)==1 % if each row just has one unknown
            dummyMat(i,decodePsb)=xor(encoded_m(i),...
                                      dummyMat(i,decodeM(1)));
            for j=2:length(decodeM)
                dummyMat(i,decodePsb)=xor(dummyMat(i,decodePsb),...
                                      dummyMat(i,decodeM(j)));
            end
            if decodedData(decodePsb)==-1 || decodedData(decodePsb)...
                                        ==dummyMat(i,decodePsb)
                decodedData(decodePsb)=dummyMat(i,decodePsb);
            else
                dummyMat
                error('decoding data does not agree!!!')
            end
            changed = 1;
```

```
52              end
53          end
54  end
```

```
function [changed,dummyMat,encoded_m,decodedData newChangedRows index]...
    = decode_round1f(changed,dummyMat,encoded_m,decodedData,...
      degreeMappingVec,changedRows,index)
% Decode round 1 for the dummyMat after the first k encoded packets
% i.e. FOLLOWING_CALL
%
% Comment added      16/March/2013    by Sasha Wang

newChangedRows = zeros(1,10);
assert(index==1,'index should be 1!')

%% Decoding Round 1 ---
% (Simple Decoding) dealing with rows with one unknown

% check if the newly arrived packet has a degree 1
len = length(encoded_m);
if degreeMappingVec(len)==1 % if there is only one unknown link
                            % to a packet
    decodePsb = find(dummyMat(len,:) == -1);
    if ~isempty(decodePsb)
        % replace the degree one position with the enocode packet
        dummyMat(len,decodePsb)=encoded_m(len);
        newChangedRows(index) = len;
        index = index + 1;
        % This is a check for either the about to be changed
        % decodedData position contains either -1(haven't decoded) or
        % the same value as it been previously decoded
        if decodedData(decodePsb)==-1 || ...
                decodedData(decodePsb)==encoded_m(len)
            decodedData(decodePsb)=encoded_m(len);
            % decodePack records the position of the newly
            % knownPacket(first element) and the value (second element).
            %                 decodedPackInfo=[decodePsb,encoded_m(i)];
        else
            dummyMat
            decodedData(decodePsb)
            encoded_m(len)
            error('decoding data does not agree!!!')
        end
        changed = 1;
    end
end



for ind=1:length(changedRows)
    i = changedRows(ind);
    if i == 0
        break
    end
```

```
52
53      decodeM = find(dummyMat(i,:) == 1 | dummyMat(i,:)==0);
54      if ~isempty(decodeM)
55          decodePsb = find(dummyMat(i,:) == -1);
56          % if each row just has one unknown
57          if length(decodePsb)==1
58              dummyMat(i,decodePsb)=xor(encoded_m(i),...
59                                        dummyMat(i,decodeM(1)));
60              for j=2:length(decodeM)
61                  dummyMat(i,decodePsb)=xor(dummyMat(i,decodePsb),...
62                                            dummyMat(i,decodeM(j)));
63              end
64              newChangedRows(index) = i;
65              index = index + 1;
66              if decodedData(decodePsb)==-1 || ...
67                      decodedData(decodePsb)==dummyMat(i,decodePsb)
68                  decodedData(decodePsb)=dummyMat(i,decodePsb);
69              else
70                  dummyMat
71                  error('decoding data does not agree!!!')
72              end
73              changed = 1;
74          end
75      end
76  end
```

```
function [changed dummyMat decodedData changedRows index arr]=...
    decode_round2i(encoded_m,decodedData,degreeMappingVec,...
    dummyMat,changedRows,index,arr,changed)
% Decode round 2 for the first k encoded packets i.e. INITIAL_CALL
%
% Comment added     16/March/2013    by Sasha Wang

for i = 2:length(arr)
    if ~isempty(arr{i}) && ~isempty(arr{i-1})
        % if both the current and the prev is not empty
        for j = 1:length(arr{i})
            for k = 1:length(arr{i-1})
                curr_row = arr{i}(j);
                prev_row = arr{i-1}(k);
                diff_pos = find(dummyMat(curr_row,:) ~= ...
                                        dummyMat(prev_row,:));
                if length(diff_pos) == 1 && ...
                        dummyMat(curr_row,diff_pos) == -1
                    dummyMat(curr_row,diff_pos) = ...
                        xor(encoded_m(curr_row),encoded_m(prev_row));
                    changedRows(index) = curr_row;
                    index = index + 1;
                    if decodedData(diff_pos) == -1 || ...
                            decodedData(curr_row) == ...
                                dummyMat(curr_row,diff_pos)
                        decodedData(diff_pos) = ...
                            dummyMat(curr_row,diff_pos);
                    end
                    changed = 1;
                end
            end
        end
    end
end
```

```
function [changed dummyMat decodedData changedRows index arr]=...
    decode_round2f(encoded_m,decodedData,degreeMappingVec,...
                   dummyMat,changedRows,index,arr,changed)
% Decode round 1 for the dummyMat after the first k encoded packets
% i.e. FOLLOWING_CALL
%
% Comment added      16/March/2013    by Sasha Wang

assert(length(encoded_m) == length(degreeMappingVec),...
            'length of encoded_m and degree array does not match!')

if length(encoded_m) ~= size(dummyMat,1)
    encoded_m_len = length(encoded_m)
    dummyMat_row_len = size(dummyMat,1)
end
assert(length(encoded_m) == size(dummyMat,1),...
            'length of encoded_m and dummyMat size does not match!')


arr{degreeMappingVec(end)}(end+1) = length(encoded_m);

curr_row = length(encoded_m);

for i = 1:length(arr{degreeMappingVec(end)-1})
    less1_row = arr{degreeMappingVec(end)-1}(i);
    diff_pos = find(dummyMat(curr_row,:) ~= dummyMat(less1_row,:));
    if length(diff_pos) == 1 && dummyMat(curr_row,diff_pos) == -1
        dummyMat(curr_row,diff_pos) = xor...
            (encoded_m(curr_row),encoded_m(less1_row));
        changedRows(index) = curr_row;
        index = index + 1;
        if decodedData(diff_pos) == -1 || ...
                decodedData(diff_pos) == dummyMat(curr_row,diff_pos)
            decodedData(diff_pos) = dummyMat(curr_row,diff_pos);
        end
        changed = 1;
    end
end
```

Bibliography

[1] 3GPP, "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs," TS 26.346, 3rd Generation Partnership Project (3GPP), 2012.

[2] S. Aoki, T. Shimizu, S. Nishimura, and K. Aoki, "Efficient Reliable Multicast Using FEC and Limited Retransmission for HDTV IP Broadcasting," *Consumer Communications and Networking Conference*, pp. 813–817, 2008.

[3] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based Erasure-resilient Coding Scheme," 1995.

[4] E. A. Bodine and M. K. Cheng, "Characterization of Luby Transform Codes with Small Message Size for Low-latency Decoding," *ICC '08 IEEE International Conference on Communications*, 2008.

[5] J. W. Byers, M. Luby, and M. Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1528 –1540, 2002.

[6] P. Cataldi, M. P. Shatarski, M. Grangetto, and E. Magli, "Implementation and Performance Evaluation of LT and Raptor Codes for Multimedia Applications," *2006 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2006.

[7] Z. Chen and Q. Zhou, "Implementation of LT Codes with a Revised Robust Soliton Distribution by Using Kent Chaotic Map," *2010 International Workshop on Chaos-Fractal Theory and its Applications*, 2010.

[8] D.J.C.MacKay, "Fountain Codes," *IEE Proc.-Commun.*, vol. 152, pp. 1062–1068, Dec. 2005.

[9] DVB Project Office, "DVB Fact Sheet: 2nd Generation Terrestrial," tech. rep., Feb 2013.

[10] Elias, P., "Coding for Two Noisy Channels," in *Information Theory, Third London Symposium*, pp. 61–76, London, England, 1955.

[11] E. Elliott, “Estimates of Error Rates for Codes on Burst-noise Channels,” *Bell Syst. Tech. J*, vol. 42, no. 9, pp. 1977–1997, 1963.

[12] ETSI DVB TM-CBMS1167, “IP Datacast over DVB-H: Content Delivery Protocols,” 2005.

[13] G. Faria, J. Henriksson, E. Stare, and P. Talmola, “DVB-H: Digital Broadcast Services to Handheld Devices,” *Proceedings of the IEEE*, vol. 94, pp. 194–209, 2006.

[14] E. N. Gilbert *et al.*, “Capacity of a Burst-noise Channel,” *Bell Syst. Tech. J*, vol. 39, no. 9, pp. 1253–1265, 1960.

[15] D. Gomez-Barquero, P. Gomez, D. Gozalvez, B.Sayadi, and L.Roullet, “Base Band inter-frame FEC (BB-iFEC) for Next Generation Handheld DVB-NGH,” *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2011.

[16] D. Gozalvez, D. Gomez-Barquero, and N. Cardona, “Performance Evaluation of the MPE-iFEC Sliding RS Encoding for DVB-H Streaming Services,” *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2008.

[17] E. Hyytia, T. Tirronen, and J. Virtamo, “Optimal Degree Distribution for LT Codes with Small Message Length,” *INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, 2007.

[18] G. Joshi, J. B. Rhim, J. Sun, and D. Wang, “Fountain Codes,” tech. rep., Massachusetts Institute of Technology, Dec 2010.

[19] K. Kwon, K. H. Im, and J. Heo, “An Improved FEC System for Next Generation Terrestrial 3D HDTV Broadcasting,” *2012 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 327–328, 2012.

[20] Z. Li and T. Herfet, “MAC Layer Multicast Error Control for IPTV in Wireless LANs,” *IEEE Transactions on Broadcasting*, vol. 55, pp. 353–362, Jun 2009.

[21] S. Lin and D. J. Costello, *Error Control Coding*. Pearson Prentice Hall, 2 ed., 2004.

[22] F. Lu, C. H. Foh, J. Cai, and L. T. Chia, “LT Codes Decoding: Design and Analysis,” *ISIT*, 2009.

[23] M. Luby, "LT Codes," *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 271–280, Nov. 2002.

[24] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu, "Raptor Codes for Reliable Download Delivery in Wireless Broadcast Systems," in *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 1, pp. 192–197, IEEE, 2006.

[25] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of Random Processes via And-or Tree Evaluation," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pp. 364–373, Society for Industrial and Applied Mathematics, 1998.

[26] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-resilient Codes," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 150–159, ACM, 1997.

[27] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient Erasure Correcting Codes," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 569–584, 2001.

[28] D. J. MacKay, *Information Theory, Inference and Learning algorithms*. Cambridge university press, 2003.

[29] F. MacWilliams and N. Sloane, *The Theory of Error-correcting Codes*. North-Holland, 1996.

[30] MathWorks, "Bit Error Rate for Coded AWGN Channels." `http://www.mathworks.com/help/comm/ref/bercoding.html`.

[31] MathWorks, "Detect and Correct Errors in a Reed-Solomon Code Using MATLAB." `www.mathworks.com/help/comm/ug/error-detection-and-correction.html`.

[32] MathWorks, "Reed-Solomon Decoder." `www.mathworks.com/help/comm/ref/rsdec.html`.

[33] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*. Wiley, 2 ed., 2006.

[34] J. Nonnenmacher and E. W. Biersack, "Reliable Multicast: Where to Use Forward Error Correction," *IFIP 5th International Workshop Protocols High-Speed Networks*, pp. 134–148, 1996.

[35] D. Plets, W. Joseph, L. Verloock, E. Tanghe, L. Martens, E. Deventer, and H. Gauderis, "Influence of Reception Condition, MPE-FEC Rate and Modulation Scheme on Performance of DVB-H," *IEEE Transactions on Broadcasting*, vol. 54, pp. 590–598, 2008.

[36] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[37] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, 1997.

[38] B. Sayadi, Y. Leprovost, S. Kerboeuf, M. L. Alberi-Morel, and L. Roullet, "MPE-IFEC: An Enhanced Burst Error Protection for DVB-SH Systems," *Bell Lab. Tech. J.*, vol. 14, pp. 25–40, May 2009.

[39] E. Schooler, J. Gemmell, *et al.*, "Using Multicast FEC to Solve the Midnight Madness Problem," tech. rep., Citeseer, 1997.

[40] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[41] A. Shokrollahi, "Raptor Codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[42] A. Shokrollahi, "The Development of Raptor Codes," Jan. 2011. Invited talk at the Kungliga Tekniska hogskolan.

[43] B. Sklar, "Reed-Solomon Codes." `http://hscc.cs.nthu.edu.tw/~sheujp/lecture_note/rs.pdf`.

[44] C. Studholme and I. Blake, "Windowed Erasure Codes," *2006 IEEE International Symposium on Information Theory*, pp. 509–513, 2006.

[45] G. Tan and T. Herfet, “Application Layer Hybrid Error Correction with Reed-Solomon Code for DVB Services over Wireless LANs,” *International Conference on wireless Communications, Networking and Mobile Computing*, pp. 2952–2955, 2007.

[46] H. S. Wang and N. Moayeri, “Finite-state Markov Channel – a Useful Model for Radio Communication Channels,” *Vehicular Technology, IEEE Transactions on*, vol. 44, no. 1, pp. 163–171, 1995.

[47] X. Wang, A. Willig, and G. Woodward, “Improving Fountain Codes for Short Message Lengths by Adding Memory,” *IEEE 8th International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2013.

[48] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, vol. 1. Prentice hall New Jersey, 1995.

[49] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes*. IEEE Press, 1994.

[50] A. Willig, “Performance Evalutation Techniques Statistics Refresher,” tech. rep., Telecommunication Networks Group (TKN) Technical University Berlin, June 2005.

[51] A. Willig, “Evaluation of Results and Run-Length Control,” tech. rep., Telecommunication Networks Group (TKN) Technical University Berlin, July 2007.

[52] J. K. Zao, M. Hornansky, and P. L. Diao, “Design of Optimal Short-Length LT Codes Using Evolution Strategies,” *2012 IEEE World Congress on Computational Intelligence (WCCI)*, pp. 1–9, 2012.

[53] H. Zhu, G. Zhang, and G. Li, “A Novel Degree Distribution Algorithm of LT Codes,” *2008 11th IEEE International Conference on Communication Technology Proceedings*, 2008.